MBC-4000 SERIES
SOFTWARE ENCYCLOPEDIA
AND
USER'S GUIDE

SANYO

FCC NOTICE

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interface when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

PREFACE

Please read this manual thoroughly before you begin to install and use Sanyo's 4000 series computers.

All rights reserved. No part of this manual may be produced in any form without express written permission from Sanyo Business Systems Corporation, Computer Division, 51 Joseph Street, Moonachie, New Jersey 07074.

The contents of this manual are subject to change. If you wish to receive revisions and updates, please complete the attached reply card.

Sanyo Business Systems Corporation assumes no responsibilities for errors in this manual or their consequences.

CP/M-86 is the registered trademark of Digital Research.

Name of Manual
Number of Manual (on back cover)
Computer Model [] 4000 [] 4050

Name
Street
City State Zip

Date purchased
What will the computer be used for [] Business Type
[] Personal
[] Educational
[] R & D
[] Other
Where did you hear about Sanyo []Advertisement
[]Literature (magazines)
[]Store Demonstration
[]Recommendation of a friend
Thank you for purchasing a Sanyo Computer.

INTRODUCTION

The Sanyo 4000 Series computers are general purpose, low cost 16 bit desk top computers. They feature high performance hardware and employ the CP/M-86 operating system.

The 4000 Series features the CP/M-86 operating system complete with editor, and utilities. The 128K RAM memory may be expanded up to 512K. 640K per drive is available.

TABLE OF CONTENTS

CHAPTER 1 GETTING STARTED 1-1
Recommended Diskettes 1-1
INSTALLATION 1-2
Environment 1-2
System Connection 1-2
Printer Connection(optional) 1-3
NAMES AND FUNCTIONS OF THE PARTS 1-4

CHAPTER 2 OPERATION 2-1
Inserting the Diskette 2-1
Keyboard 2-2
Disk Formatting 2-5
Copying Disks 2-5
Running the Operating System 2-7

CHAPTER 3 IPSO FUNCTIONS--INPUT/PROCESS/STORE/OUTPUT 3-1
Input 3-1
Process 3-1
Store 3-2
Output 3-2
CHAPTER 4 - COMPUTER VOCABULARY 4-1
Vocabulary 4-1

CHAPTER 5 - PROGRAMMING AND FLOW RELATIONSHIPS, OPERATING SYSTEMS 5-1
Instruction and Form. 5-1
The Operating System 5-2
Flow 5-2

CHAPTER 6 - INTRODUCTION TO CP/M-86 6-1
Bringing Up CP/M-86 6-1
Help with CP/M-86 6-1
Working with Drive B 6-1
Typing in the File Name or Command 6-2
Correction of Entry 6-2
Examination of the Directory 6-3
File Names 6-3
Use of the Type Command 6-3
Use of the REN (RENAME) Command 6-4
Use of the ERA (ERASE) Command 6-4
Printer Interaction 6-5
Use of PIP 6-5
COPYING FILES. 6-5
COPYING ALL FILES 6-6
MERGING FILES TOGETHER 6-6
COPY TO OTHER DEVICES 6-7
SPECIAL PARAMETERS DURING COPY AND TRANSFER 6-8
Use of LDCOPY 6-9

Use of the STAT Command 6-10
Use of the ED Command 6-12
Use of the ASM86 Command 6-13
Use of the DDUMP Command 6-13
Use of the DDT86 Command 6-13
Use of the SUBMIT Command 6-15
Use of the MOVCPM Command 6-15
Specifying Files with Commands 6-15
File Referencing 6-16
Drive Specification 6-17
Number of Files on a Diskette 6-17
COMMANDS 6-19
DIR 6-19
ERA 6-20
FORMAT 6-21
REN 6-22
STAT 6-23
SUBMIT 6-27
TYPE 6-29
PIP 6-30
Other CP/M-86 Data 6-33

CHAPTER 7- PREFACE TO SANYO-BASIC PROGRAMMING 7-1
History 7-1
Setup 7-1
Display 7-2
Syntax 7-3
Listing of Commands 7-3
Running a Program 7-3
Clearing a Program. 7-4
Ending a Program. 7-4
Variables 7-5
Arrays. 7-6
Transfer of a Program Execution 7-7
Summary 7-8

CHAPTER 8 - MATH AND LOGICAL RELATIONSHIPS 8-1
Mathematical Operators. 8-1
Use of Parentheses in Expressions 8-2
Priority of Sequence in Operator Processing 8-2
Relational Operators. 8-3
Logical Operators 8-4
Operators Onto Strings. 8-6
Accuracy in Expressions 8-6

CHAPTER 9 - SANYO-BASIC 9-1
Symbols for Syntax Definition 9-1
Operation Modes of SANYO-BASIC 9-2
INITIALIZATION 9-2
PROMPT AND CURSOR 9-3
DIRECT MODE. 9-3
PROGRAM EDIT MODE 9-4

Index Continued

RUN MODE 9-4
RETURN TO OPERATING SYSTEM MODE 9-4
Line and Statement Use in a Program 9-5
LINES AND LINE NUMBERS 9-6
KEYWORDS AND SPACES 9-7
DATA 9-7
Expressions and Operators 9-11
ARITHMETIC OPERATORS 9-12
ARITHMETIC EXPRESSIONS 9-13
RELATIONAL OPERATORS 9-13
RELATIONAL EXPRESSION 9-14
LOGICAL OPERATORS 9-15
LOGICAL EXPRESSION 9-15
EVALUATION OF COMBINED EXPRESSIONS 9-18
STRING EXPRESSION 9-19
INSTRUCTION WORDS 9-19

CHAPTER 10 - FUNCTIONS 10-1
Explanatory Symbols 10-2

CHAPTER 11 - PROGRAM EDITING. 11-1
Line Edit Mode. 11-1

CHAPTER 12 - DISK FILES 12-1
Types of File 12-1
Filenames 12-1
Number of Files 12-3
Instructions for Files. 12-3
Program Files 12-4
Data File 12-4
Sequential Files. 12-4
Random Files. 12-7
Index Sequential Files 12-8
Input/Output of Files and Disk Images 12-11

CHAPTER 13 - DISK UTILITY PROGRAMS AND THEIR USE 13-1
Copying Disks 13-1
File Transfer Program 13-3
Procedure to Format a Diskette. 13-5
Procedure to Copy the Operating System. 13-7
Procedure for Copying All the Files from A to B 13-8
Sort Utility. 13-9
SEQUENCE OF OPERATIONS 13-10
ERROR MESSAGES FOR THE SORT UTILITY. 13-17
Merge Utility 13-18
FDUMP Utility 13-21
XREF Utility. 13-22
XCHG Utility. 13-23
COPYDISK Utility. 13-24
DDUMP Utility 13-25
ERROR DISPLAY 13-28

Index Continued

CHAPTER 14 - DIAGNOSTICS 14-1
Trouble-Shooting List and Remedies 14-1

CHAPTER 15 - INPUT / OUTPUT SYSTEMS AND CONNECTORS 15-1
The Printer Connector 15-1
The Serial Port 15-1
The Floppy Disk Interface 15-2
The Keyboard. 15-2
Configuration Control of the RS-232C Port 15-3

CHAPTER 16 - INPUT / OUTPUT SOFTWARE AWARENESS 16-1
Peripheral Use. 16-1

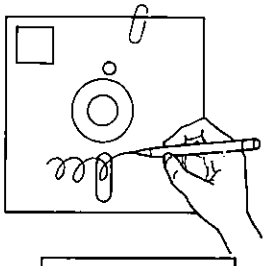
CHAPTER 17 - SPECIFICATIONS. 17-1
Hardware. 17-1
Display 17-1
Software 17-2
HARDWARE CONFIGURATION 17-3
Connector Pins 17-4
Memory Map for Main CPU and Sub CPU 17-5
I/O Map 17-6
BIOS Interface 17-7
Use of RS232 Serial Port 17-8
Command and Status Formats 17-9
Changing Baud Rate 17-10
Example of RS232 Program 17-11
Simple Trouble-shooting 17-12

TECHNICAL REFERENCE R-1
List of Reserved Words R-2
Character Codes for the Sanyo. R-3
ERROR MESSAGES R-4
Memory Maps. R-8
MBC-4000 Series Use of the RS-232-C Port. R-11
The RS-232C Input/Output Serial Port R-14
The Most Common RS-232-C Circuits R-15
TYPE OF SIGNAL. R-17
Control and ESC Sequence Codes R-19
Escape Sequence (Cursor Addressing) R-20

CHAPTER 1 GETTING STARTED

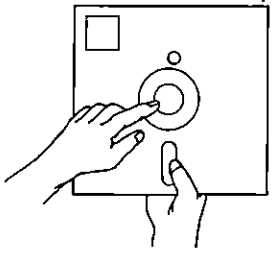
Diskette Handling

(a)



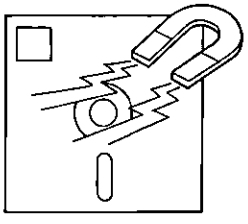
Do not pin or clip the diskette jackets. Do not write anywhere on the disk except on the label. Always write with a felt tip pen; do not use a pencil or ink pen.

(b)



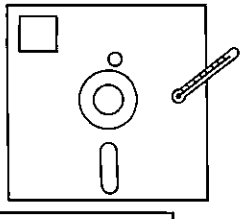
Do not touch the magnetic recording face. Never clean with alcohol or similar material.

(c)



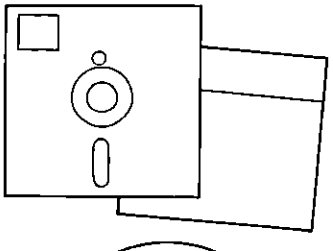
Never expose disks to magnetic or electric fields.

(d)



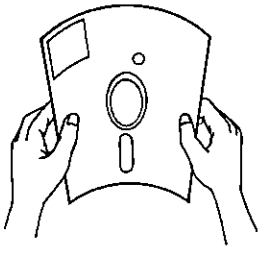
Do not store disks in direct sunlight, extreme heat or cold.

(e)



Keep the diskettes inside their envelopes to protect against dust or dirt.

(f)



Do not bend or fold diskettes.

<u>Always</u> make copies of original disks for safe keeping.

Recommended Diskettes

Sanyo recommends the use of the MD2DD diskettes (double sided, double density, soft sectored). Sanyo Computers assume no responsibility for trouble arising from the use of other diskette types.

INSTALLATION

Environment

Do not place your computer in an area exposed to extreme temperatures (i.e. in direct sun, near heaters or air conditioners).

Do not place close to a wall; allow for passage of air around the computer.

Avoid dusty areas and areas exposed to vibrations.

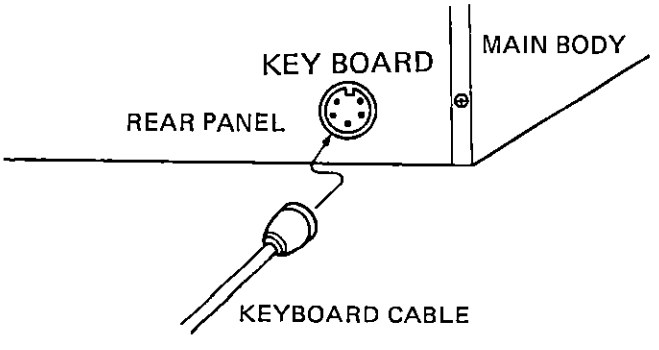
Use local AC power outlets. Remove diskettes before switching the power on or off.

Do not remove the computer cabinet and tamper with the internal mechanisms.

System Connection

Please read the following instructions before connecting the system. Check each box [] as you complete each step.

Connect the keyboard as shown below.

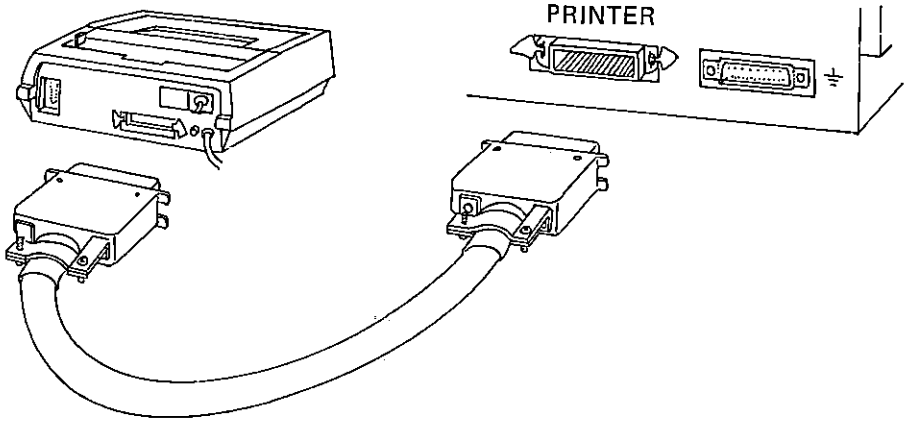


[] Remove the vinyl cover the plugs and connectors.

[] Join the keyboard cable to the computer socket labeled KEY BOARD.

Printer Connection (optional)

Connect the printer as shown below.

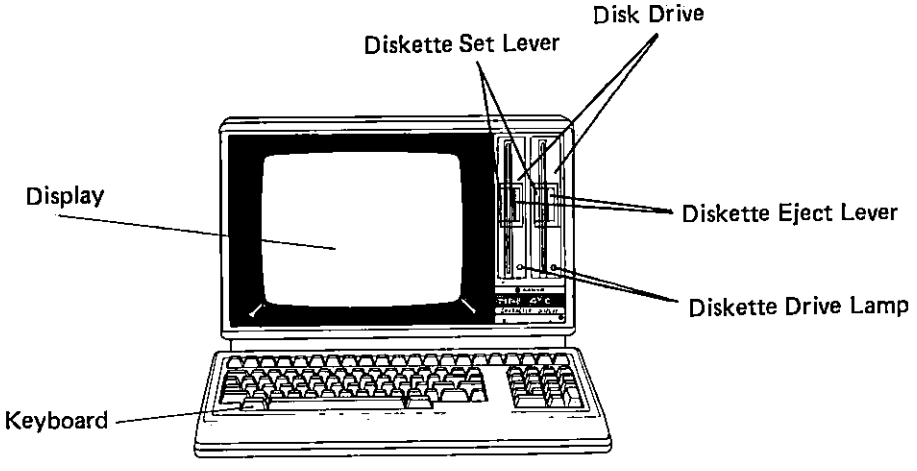


[] Connect the printer cable to the computer connector labeled PRINTER. Lock by pressing springs into the slots on the printer plug.

[] Connect the printer cable to the printer.

NAMES AND FUNCTIONS OF THE PARTS

Names



1 Display

The twelve inch CRT displays alphanumeric characters, eighty characters across on twenty four lines. Never clean the screen surface with thinner or similar material as it might damage its non-glare coating.

2 Disk Drive

This is a built-in 5 1/4 inch double density, double track drive. One more drive may be added.

3 Diskette Set Lever

Insert the diskette into the drive and move the lever right to set the diskette.

4 Diskette Eject Lever

This lever releases the diskette set lever. Press it in as you insert a diskette. Press it in after the Set Lever has been used to eject a diskette.

5 Diskette Drive Lamp

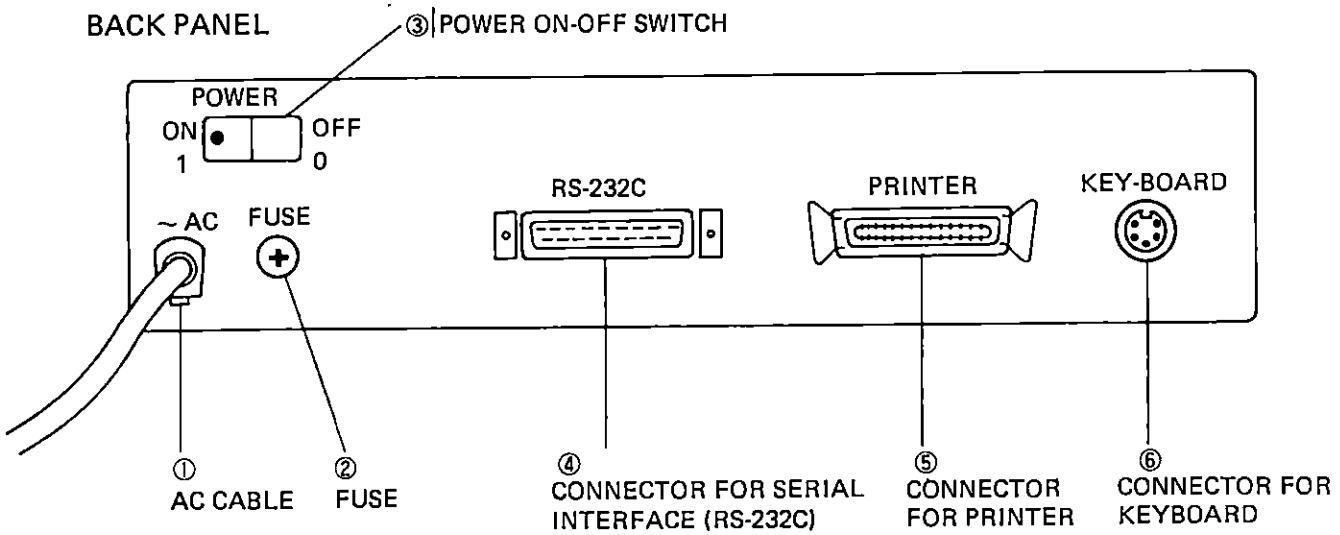
This lamp lights to indicate the disk drive being used.

6 Power On Lamp

This lamp lights when the computer has power.

7 Keyboard

The keyboard contains fifteen function keys, cursor control keys, numeric keys, ten key pad, math function keys, and editing keys in ASCII specifications.


BACK PANEL
③ POWER ON-OFF SWITCH
POWER
ON
1
OFF
0
~ AC
FUSE
RS-232C
PRINTER
KEY-BOARD
① AC CABLE
② FUSE
④ CONNECTOR FOR SERIAL INTERFACE (RS-232C)
⑤ CONNECTOR FOR PRINTER
⑥ CONNECTOR FOR KEYBOARD

1 Power On/Off Switch

If this switch is turned on, the Power On Lamp is lit and operations may begin. If it is turned off, power is terminated and operations will cease. The Power On Lamp will be extinguish.

If the power is turned off, everything will be erased in internal memory. Copy the memory contents on diskettes before turning the power off.

2 AC Cable

This connects the computer to the AC outlet (50/60Hz).

3 FG Terminal

An electrical ground is attached here.

4 Fuses

Fuses are provided to protect the internal circuits. If you must change a fuse, use only a two (2) ampere fuse. Turn the power off.

5 Serial Interface Connector (RS-232)

This connects an external printer to the computer.

6 Printer Connector

This connects a printer to the computer.

7 Keyboard Connector

This connects the keyboard with the computer.

8 Brightness Control

When the brightness control is turned clockwise,the screen becomes bright; when turned counterclockwise, the screen brightness lessens gradually . If the screen does not display anything when power is on, the brightness control may be turned too far to the left.

9 V-HOLD Control

This controls the vertical timing. It is adjusted for 60HZ at the time of delivery.

10 Ventilation Slots

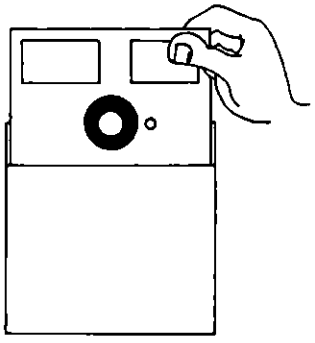
These are cooling slots for the computer.
Never cover these slots.

CHAPTER 2 OPERATION

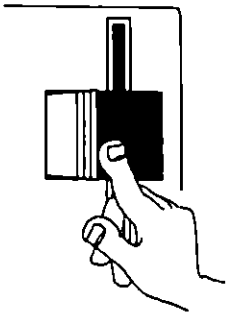
Disk Drive

Never insert or withdraw a disk while the drive is running. The input symbols, A> or B> for DOS mode and > for SANYO BASIC mode will be displayed when the drive unit is not running.

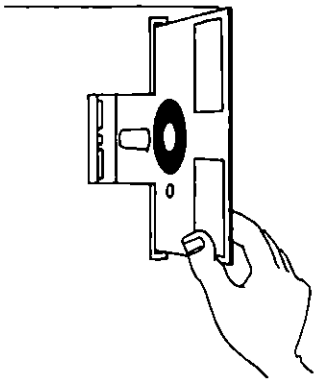
Inserting the Diskette



Withdraw the diskette from its envelope.



Open the drive door by moving the gate lever left.



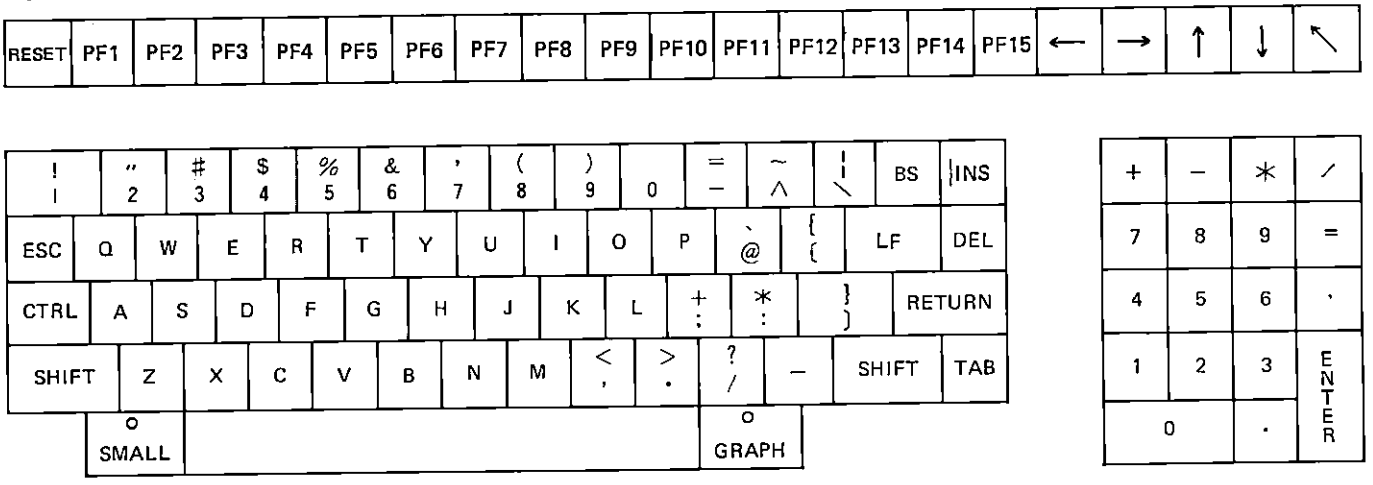
Hold the diskette by the label and insert into drive. The write protect cut out on the disk edge will face down and the read/write slot will lead into the drive. Insert the diskette till the end of its passage.

Close the door by moving the gate lever right.

Withdrawing the Diskette

Open the drive door by moving the gate lever left. Withdraw the diskeete and close the door by moving the diskette set lever right.

Keyboard


RESET PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8 PF9 PF10 PF11 PF12 PF13 PF14 PF15 ← → ↑ ↓ ↖
! 1 " 2 # 3 $ 4 % 5 & 6 ' 7 (8) 9 0 = − ~ ^ | \ BS INS
ESC Q W E R T Y U I O P ` @ { [LF DEL
CTRL A S D F G H J K L + ; * : }] RETURN
SHIFT Z X C V B N M < , > . ? / _ SHIFT TAB
SMALL
GRAPH
+ − * /
7 8 9 =
4 5 6 ,
1 2 3 ENTER
0 .

Shift Display

The keys are imprinted with only the upercase characters they represent. They can be used to input commands in both uppercase and lowercase.

When a letter key is depressed, the corresponding uppercase letter will be input.

To input lowercase letters, depress the SMALL key. The Lowercase Lamp in the key glows, indicating the ability to input lowercase letters with a single key depression. While the lamp is lit, uppercase letters are then obtained by first depressing the SHIFT key, then the letter key.

By depressing the SMALL key again, uppercase letters are input directly by single key depression.

When the Lowercase Lamp is out, lowercase letters can only be obtained by first depressing the SHIFT key, then the letter key.

Repeat

The keyboard is equipped with the auto-repeat function. If a key is depressed for about 0.5 seconds, that character will be input again and again.

[SMALL]

If the SMALL key is depressed and its lamp lights, lowercase alphabetic characters will be input. Uppercase letters can only be obtained by depressing the SHIFT key then the letter key.

[GRAPH]

If the GRAPH key is depressed in SBASIC, a beep sounds and the Graph Lamp in the key lights indicating that graphic mode is on. The keys on the keyboard now represent graphic characters and may be input.

To exit the graphic mode, depress the GRAPH key again. A beep sounds and the Graph Lamp goes out.

[PF 1]
.
.
.
[PF 15]

Each function key may be assigned a character string while in SBASIC. Depressing the function key then will have the same effect as entering successively the character keys corresponding to the string.

In the DOS mode, CP/M transient commands are assigned to these keys

PF 1	DIR
PF 2	DIR B:
PF 3	REN
PF 4	SAVE
PF 5	ERA
PF 6	STAT
PF 7	SUBMIT
PF 8	PIP

PF 9 ED
PF 10 ASM86
PF 11 GENCMD
PF 12 DDT86

SOFTWARE SUPPORTED KEYS:

PF 13 Cursor Right (ESC C)
PF 14 Cursor Up (ESC A)
PF15 Direct Cursor Addressing(ESC F)

Numeric Keys (Ten Key Pad)	These keys are intended for business ooperations. The SHIFT, SMALL, and GRAPH keys do not effect their operation.
Edit Keys	These "arrow" keys move the cursor.
[RESET]	This key is depressed together with the CTRL key to reset the system.
[CTRL]	This key changes the system to control mode.
[TAB]	The TAB key shifts characters eight positions to the right when depressed in the DOS mode.
[DEL]	The DEL key erases the character to the left of the cursor and displays the erased character to the right of the cursor in the DOS mode. In basic mode, the DEL key erases the character to the left of the cursor.
[BS]	The back space key erases the character to the left of the cursor as it moves left.
[LF]	The line feed key checks for valid commands and starts new lines in the DOS mode.
[RETURN]	The RETURN key checks for valid commands and returns the cursor to home position. It starts new lines in basic mode.
[ESC]	This key is used to create the escape sequence in the basic mode.
[ENTER]	This key has the same function as the RETURN key.
[INS]	This is the key to insert a space at the cursor position in a BASIC mode. This key does not have an auto repeat function.

Disk Formatting

All blank diskettes must be Sanyo formatted before being used. This allows your Sanyo computer to write and read information on a familar disk. Depress the RETURN Key after each entry in the steps below.

Step 1 Insert a system disk in drive A. When the system input symbol is displayed (A>), enter the command FORMAT. The following prompt will be displayed:

```
FORMAT VER X.X
INSERT NEW DISK IN DRIVE A, THEN PRESS Y
(OR CTRL-C TO REBOOT)
```

Step 2 Withdraw the system disk and insert the blank disk, then enter a capital Y. This will start the formatting process.

Step 3 The following prompts will be displayed:

```
DISK TEST PROGRAM RUNNING
WAIT FOR A FEW MINUTES
```

The diskette will now be tested for correct formatting. This will take about two minutes.

Formatting is completely finished when the the following prompt is displayed:

```
DISK TEST COMPLETED
NO ERRORS
```

You may format another diskette when the "INSERT NEW DISK ....." prompt is displayed.

CAUTION! Formatting a diskette containing programs and data will erase all these contents. Be sure to take the system disk out before entering the Y for the start of the formatting process. Do not format a diskette with applications, programs or data on it.

If you wish only to test a diskette for correct formatting, enter a T at the prompt. You may exit the formatting program by depressing the CTRL Key and the C Key at the same time (^C).

Copying Disks

It is always a good habit to copy master diskettes and diskettes with important data on them. This backup avoids problems that may arise if you mistakenly destroy programs or data on a diskette.

Single Drive

Step 1 If you have a single drive, you will use the Filecopy Utility. When the system input symbol is displayed (A>), enter FILECOPY and the file name plus qualifier of the file you wish to copy.

The following prompts will be displayed:

```
SINGLE DRIVE FILECOPY VER-X.X
ENTER FILE NAME(OR PRESS <CTRL-C> TO REBOOT)
```

Step 2 Enter the file name to be copied,then depress the RETURN key. The following prompt will be displayed:

```
INSERT SOURCE DISK THEN PRESS <RETURN>
```

Step 3 Place the disk which contains the file you wish to copy in the drive and depress the RETURN key. The following prompt will be displayed:

```
INSERT DESTINATION DISK THEN PRESS <RETURN>
```

Step 4 Place the disk on which the file is copied in the drive and depress hte RETURN key. When the file is copied, the following prompt will be displayed:

```
FILE COPY COMPLETED
ENTER FILE NAME(OR PRESS <CTRL-C> TO REBOOT)
```

Depress the CTRL-C keys and you will be placed back into the DOS mode providing the disk in the drive has CP/M on it. If it doesn't, insert the original disk or a CP/M disk in the drive before the RETURN key is depressed.

Dual Drives

Step 1 If you have two drives you may use the COPYDISK Utility. This disk copy utility allows you to copy the entire disk in one command.

Depress the RETURN Key after each entry.

When the system input symbol (A>) is displayed, enter COPYDISK. The following prompt and instructions are displayed:

```
CP/M-86 Full Disk COPY Utility Version 2.0
Enter Source Disk Drive (A-D)?
Destination Disk Drive (A-D)?
```

Enter the source drive name (the name of the drive the disk is on that you want copied). The destination disk drive contains the blank disk.

The following prompt is displayed after you enter the drive names:

```
Copying Disk A: to Disk B:
Is this what you want to do (Y/N)?
```

You have the opportunity of exiting form the disk copy utility if you enter an N. If you wish to continue, enter Y .

The following prompts appear:

```
Copy started
Reading Track XXX
```

```
Writing Track XXX
Verifying Track XXX
```

These last three prompts will appear until the entire disk is copied.

You may copy another disk when the disk copying instructions are displayed again.

File Copying

The PIP command allows you to copy files instead of an entire disk.

Step 1 When the system input symbol (A>) is displayed, enter PIP ^X1:= X2: file name.qualifier where X1: is the the drive where the files are written to, and X2: is the source.

Example: File TEST.ONE is on a disk in drive A. You want to copy this file to a disk in drive B. Enter the following PIP B:=A:TEST.ONE . Depress the RETURN Key.

The file, TEST.ONE is now copied from the disk on drive A to the disk on drive B.

The PIP command (PIP.COM) must be present on the disk that a file is copied from.

Refer to the Command section of this manual for further information on the PIP command.

Running the Operating System

The following prompts will appear after you insert a system disk and close the door:

```
CP/M-86 version x.x
Segment Address = 00040
Last Offset     = 357F

RAM is available for 00000 to 1FFFF
Bios-86 #x.x
```

This is referred to as a cold start. You are now in the DOS (Disk Operating System). DOS consists of resident programs that supervise the diskette files and run the whole system. Resident programs are always in the operating system. Transient programs may be added on according to the application.
The resident commands are:

TYPE	Displays file contents
DIR	Lists file names
REN	Changes file names
ERA	Erases file names
USER	Changes user numbers
DIRS	Displayes SYS attribute files

The transient commands are:

LDCOPY	Copies the CP/M system
ED	Corrects text file contents
PIP	Copies files and assigns names to them
ASM86	Generates hexadecimal object files
DDT86	Tests of debugs interactively programs generated under CP/M-86
SUBMIT	Executes several commands at a time
STAT	Displays device assignment
BASIC	Sanyo Basic Mode
COPYDISK	Copies entire contents of a disk
DDUMP	Displays tracks and sectors of disk
FDUMP	Displays file contents
FORMAT	Formats disks
GENCMD	Converts hex files into memory images
GENDEF	Produces custom disk parameter tables
HELP	Displays information about CP/M-86
LMCMD	Converts hex files into memory images suitable for execution under CP/M-86 accepting an Intel L-module file as input
LOADER	Moves CP/M System to memory on boot (contains LDCPM.H86,LDBIOS.A86, LDBIO.H86 and LDBIO.A86). LOADER is not used by the computer operator.
MRCV	Recieve command for downloading
MTRN	Send command for downloading
PUT	
SORT	Sorts files
TOD	Displays or sets the dateand time

These commands, followed by a file name if appropriate, are entered after the input symbol, A>.

CHAPTER 3 IPSO FUNCTIONS--INPUT/PROCESS/STORE/OUTPUT

Introduction

Your SANYO computer, and in fact all computers, have four basic functions in common. These can be called the IPSO functions. These are defined as being the input, process, store, and output functions. Let us look at each of these separately and see how they relate not only to a SANYO computer but to computers in general.

Input

No computer works just by itself, it must rely on outside information sources to provide various types of data to its internal structures. This data is usually inputted or processed into the computer. With your SANYO there are several ways information can be inputted to its internal structure.

All SANYO computers have a keyboard unit. This is the primary way you, as an operator, can process information into the system. The keyboard in the MBC series of the SANYO main frames provides an interactive element between man and machine.

Another important input port is the RS-232 or Serial Communication Port, which exists in every SANYO machine. This connector at the back provides a way for your SANYO to talk to other computers in a very fast and convenient manner. The RS-232 parameters are well defined at the end of this manual and provide a reliable and fast way for your SANYO to speak to other main frames or to devices called modems, which will allow it to talk on telephone lines. Sometimes information has to come in from a disk or other magnetic media storage device. Once again, input parameters can be processed into the SANYO through the floppy disk or other magnetic media port.

All of these different types of input help your SANYO computer understand what it has to do and what data it must work with to accomplish its task. Other forms of input are available in a peripheral sense. For example; bar code readers, which are a form of electronic reading wand, light wands, and other types of peripherals can be purchased that will allow your SANYO to accept information from written text. Most of the currently available peripherals use the RS-232 port.

Process

Your SANYO computer takes the information, which is either inputted or stored and processes it. By process we define those operations which will manipulate and transfer data in a meaningful and logical way.

The central processor of your SANYO computer is a microchip device. This extremely complex integrated circuit performs operations many millions of times each second and stands a tribute to man's technological advancement. The process function includes operations that allow your SANYO to perform arithmetic and other mathematical operations; the sorting and listing of data; the acceptance of words, characters, and texts; the storage and retrieval of important data elements and other related tasks.

The process function can be an extremely simple one or a highly

complex one, depending on the needs of the program written. The very word process defines a program as a sequence of operations. This sequence is inputted in the form of a user's program. We will look at programming later in Chapters 5 and 6, but basically a program allows a defined sequence of operations to flow. This sequence is defined by you, the user.

It is difficult to say empirically how fast the process function occurs. Many times we are in the sub-second level. During the times you watch your SANYO perform its many operations, be aware of how incredibly fast the machine is processing your information. Many times your SANYO computer will rely on other peripherals to give it information to process or it may wait in what we call a "process loop" while other peripherals process information given them by your SANYO computer. The process function is a complex one but it is one of our common IPSO parameters.

Store

Your SANYO computer, once given data to work with and having processed it in a specified way, must now place that data into areas where it can reach it conveniently. These locations are refered to as the memory of the machine and are basically storage and handling "pigeon holes" or places to hold the data that the machine is ready to process. The memory in your SANYO computer represents a block of approximately 64,000 locations which can be accessed internally by the machine. Externally the machine may have several million locations which it can access through either its RS-232 or other (IO) input and output ports.

The STORE function is bidirectional and can be allocated to be either temporary storage (as in the case of a random access memory (RAM)) or permanent storage (as read only memory (ROM)). Likewise, files which are stored outside of the machine, for example a floppy disk file, can perform the store function by saving magnetically the information sent to them.

The store function permits the central processing element to randomly select repeated blocks of information and process them, presenting them to users in an output form or to the internal workings of the machine.

Output

Having once presented information to the machine through various input devices and having stored that information either temporarily or permanently within the machine (or having processed the information) we must now concentrate on our fourth function, which is the outputting of this information.

The output of information can be in several forms. One way is in the form of magnetically stored files. In your SANYO computer this would be the floppy disk or to a hard disk type of storage media. We can use the RS-232 port for this sort of interchange to occur or through the much faster DMA (direct memory access) port which is part of the floppy disk controller. Another output port is the parallel printer port, allowing us to take information from the memories inside the SANYO and to process it for external printer devices. This form of output gives the user a hard readable copy.

An important output function is that which places data on the

screen. In your SANYO computer this is mapped into the memory structure as a display element or matrix of internal data. The CRT (cathode ray tube), also called the screen, is handled in this fashion so that very quick awareness can be shown to the user of various internal states.

In summary the output parameters are basically the way in which the computer speaks back to the user. The screen display and the printer port accomplish this function directly in user form but in machine language form the floppy disk interface accomplishes the output storage function.

Summary

We have seen in the preceding four examples how your SANYO computer completes its four basic functions of INPUT, PROCESS, STORE and OUTPUT. Every computer system must have these four functions to be an efficient machine. Many thousands of hours of engineering effort have been brought together to give your SANYO computer full realm of possibilities in each of these four important areas. The use of these areas in an efficient way defines the computer program.

CHAPTER 4 - COMPUTER VOCABULARY

Introduction

In the MBC documentations many words are used which are common to our industry. Many of these are difficult to find in dictionaries in the context of their meaning for our use. The following list is a glossary of the majority of our "special" terms.

Vocabulary

A>	CP/M prompt, waits for your entry parameter.
ADDRESS	A particular number associated with a memory location. Usually an address is a number between 0 and 65535 (or &H0000 and &HFFFF hexadecimal).
ADDRESS BUS	A set of electronic paths which carry the binary-encoded address from a microprocessor to the computer internal architectures.
ALU	Abbreviation for Arithmetic Logic Unit.
AND	A binary function that is "true" if and only if all of its input parameters are "true".
APPLICATION SOFTWARE	The collection of programs that a user can use to perform a specific task. Examples: an inventory program, a general ledger program, or a word processor.
ARRAY	A set of data elements referred to as a group. Usually referred to with a series of subscripts. For example: A 100-element array can be expressed as A(100).
ASCII	An acronym for American Standard Code for Information Interchange. This standard code assigns a unique value (from 0 to 127) to each of 128 letters, special characters, control characters, and numbers.
ASSEMBLER	A program which converts the characters (symbols) of an assembly language into machine language, generating operation code.
ASSEMBLY LANGUAGE	A language made up of mnemonics and symbols. Special groupings do special functions.
B>	CP/M prompt, waits for your entry parameter.
BASIC	An acronym for "Beginner's All-Purpose Symbolic Instruction Code". BASIC, a higher-level language, was published at Dartmouth College in the early sixties.

BAUD	Bits per second. The rate information flows between devices.
BDOS	Disk Operating System allocation for CP/M.
BINARY	A number system with but two ciphers: "0 and 1". Each digit in a binary number represents a power of two.
BINARY FUNCTION	A logical or arithmetic operation performed by a computer which has multiple inputs and only one output. Binary signals are usually used.
BIOS	I/O system allocation for CP/M.
BIT	A Binary entity that is the smallest amount of information which a computer can manipulate. A bit indicates a single value: "0" or "1". Bits can be added together to form larger words called "bytes".
BOOTSTRAP ("BOOT")	To initialize a system from a "power-off" condition. This is usually a small program that is run immediately upon power-up.
BUFFER	A register or memory which is used to hold data temporarily. Screen buffers contain information to be displayed on the video screen. Keyboard buffers hold formed input lines. Register buffers hold intermediate data.
BUS	A group of electronic paths in a computer which permit data flow from one place to another.
BYTE	A basic unit of memory in a computer's architecture. Usually comprises eight bits. Arranged in a row, bit 7 to bit 0--Bit 0 is least significant in value.
CALL	An instruction which calls a subroutine in a program, transferring execution to another location.
CHARACTER	Letters (both upper- and lower-case), numbers, various symbols, and graphic discriptors for human and computer use.
CODE	A way of expressing quantities in another form. As an example, the ASCII code represents characters as binary numbers. Languages represent math relationships in terms of program operators.
COLD-START	A start-up operation from a "power-off" state.
COMPUTER	A man-made architecture that can receive and store instructions and execute them. Computers can in-

	put, process, store, and output data.
CONCATENATE	To join or merge files or data into a new composite entity.
CONNECTOR	A physical device for attachment of conductive paths. I/O to peripheral devices is through connectors.
CONSOLE	The defining label for the immediate input and output device. Here the keyboard and the CRT.
CONSTANT	A set of data which will not change. (i.e.: The values 1, 2, 3, 4, 5, or 3.14 are constants.)
CONTROL (CTRL) CHARACTER	In the ASCII character set there are characters which have no graphic symbol and are used to control or direct various functions. A BREAK character, for example, aborts execution.
CP/M	Acronym for Control Program for Microprocessers. This label refers to the operating system that is revealed in Chapter 6.
CPU	Abbreviation for Central Processing Unit.
CRT	The "Cathode-Ray Tube", or video screen of a computer's output function.
CURSOR	A position symbol which tells an operator where display text is to be entered.
DATA	Information to, from, or in a computer.
DB25	A type of connector with 25 pins (or conductors) used for typically RS-232-C I/O communications.
DECIMAL	Usually of ten states.
DIAGNOSTIC	Usually a software program that is used to evaluate the operational performance of a machine.
DIR	CP/M Acronym for a directory list command. See Chapter 6.
DISASSEMBLER	A program which analyzes machine language and converts it into assembly language.
DISPLAY	An ouput device or peripheral of a computer, usually a video or CRT screen.
DUMP	To empty out the contents of, as in to DUMP memory to a printer or other device.
ENTRY POINT	A location used by software that contains the first

executable command in that section.

EXCUSIVE-OR	A binary function whose result is "false" only if all of its inputs are "true".
EXECUTE	A performance of a command, instruction, or software program. A program that is "RUN" is executed.
FILE	A location or storage element into which data can be placed.
FLOPPY	The jargon for the mini-diskette (both 5 1/4" and 8") for use in the disk drive I/O.
FORMAT	The physical rule or characteristic for entry or output. Can also be a verb.
FUNCTIONS	A derived sequence of events that manipulate data, variables, and constants.
GRAPHICS	The character set for video or printer output. Can also refer to special graphic characters or modes.
HARDWARE	The touchable components of any computer.
HEXADECIMAL	A numbering system which uses 0 through 9 and the six letters A through F to represent values. All hexadecimal numbers are preceded by &H.
HIGH-LEVEL LANGUAGE	A language which makes sense to human intelligence.
HIGH-ORDER	The highest value. Used in talking of bits in a byte. The high-order bit of a byte is the highest place binary value.
I/O	See Input/Output.
INDEX HOLE	Floppy disk awareness (physically) of the start of each track of information. A hole is punched in the diskette. When it passes a sensor an awareness is made.
INPUT	Data entering a computer from the outside world. It can come from a console or a remote peripheral.
INPUT/OUTPUT (I/O)	Software routines or hardware architectures which receive or transmit data with peripherals external to the computer.
INSTRUCTION	A command in a program that a computer can execute. In a high-level language, instructions may be comprised of many characters.

INTERFACE	A device or set of logics which make I/O possible.
INTERPRETER	A special program which interprets and executes a higher-level language. Most interpreters are written in machine language.
INTERRUPT	Interrupts are used to signal the computer that an external device has a status or some data for the computer to process.
K	In common use in a computer "K" usually represents 1024.
KILOBYTE	1,024 bytes.
LANGUAGE	Computer language express a communication tool between programmer and computer. SANYO-BASIC, PASCAL, FORTH, and FORTRAN are all languages.
LINE	A "line" is a horizontal grouping of graphic characters. Lines are used for both printers and in displays.
LOW-LEVEL LANGUAGE	Communication language more machine rather than human based.
LOW-ORDER	The least important element in a grouping. The low-order bit of a byte is the bit with the least binary value.
MACHINE LANGUAGE	Language for computer hardware use. Most machine languages are binary and are in hexidecimal.
MBC	A Sanyo acronym for My Business Computer.
MEMORY LOCATION	The smallest element in a memory that a computer can address. Memory locations are labeled with a unique address.
MEMORY MAP	The set of all possible memory locations a microprocessor can communicate with. Also used to describe a system's memory.
MICROCOMPUTER	A computer whose architecture is reliant upon a microprocessor.
MICROPROCESSOR	A single integrated circuit computer which performs the basic IPSO functions and executes machine language programs.
MNEMONIC	An easy way to remember something used in the place of something more difficult to remember. An abbreviation.
MODE	A state or set of conditions for which a certain

	set of rules, formats, or dates apply.
MODULO	An arithmetic function which manipulates two items.
MONITOR	1) A computer controlled television CRT or other display receiver. 2) A program which permits user access to computer architecture at the byte level.
MP/M	Acronym for Multiprogramming Control Program for Microprocessors. This label refers to the operating system that will perform internal chores for the micro-computer.
MULTIPLEXER	A device or software set with many inputs and one output. Multiplexers select one of its many data inputs to be its output.
NYBBLE	Half of a byte or four bits. Can be upper 4 or lower 4.
OPCODE	A binary machine language command.
OPERATING SYSTEM	The software architecture under which a computer communicates and functions with a human operator.
OR	A binary function whose result is "true" if at least one of its inputs is "true".
OUTPUT	Data leaving the computer into the outside world. Also refers to the process of generating same.
PAGE	A full page or screen of information on a video or CRT display.
PASCAL	A French scientist after whom an interpretive language was named.
PERIHERAL	An external device attached to the computer peripherals are input and/or output devices which communicate through the I/O.
PF	Label for the special function keys.
PINOUT	A description of the function of each pin on a connector or an IC.
PIP	CP/M acronym for a file transfer program. Stands for Peripheral Interchange Command. Allows transfer of files between devices. See Chapter 6.
PIXEL	The smallest piece or "dot" of resolution on the computer's display screen.
PRINTED CIRCUIT BOARD	The real estate for electronic circuits. Sheets of fiberglass or epoxy with copper conductors

	"etched" onto the surface. Components mount onto the traces.
PROGRAM	Sequences of commands or instructions which define a desired effect or achievement.
PROM	Acronym for "Programmable Read-Only Memory". PROMs are ROMs that can be altered. Data in PROMs is retained even if power is turned off. Some PROMs can be erased and reprogrammed.
RANDOM-ACCESS MEMORY (RAM)	Fast read-write memory for general use by a computer. The computer stores values in RAM locations and recalls them for use at any time. Data in RAM memory is lost when the power to the computer is turned off.
READ-ONLY MEMORY (ROM)	Slower memory for constant storage of programs or data for use by the computer when the power is first turned on. Information in ROMs cannot be changed. Information in ROMs is retained when the power is turned off.
RETURN	To complete a subroutine and return back to the main program that "called" it.
RS232-C	Standard for serial communications. Serial structure for I/O.
RUN	To instigate the sequence of instructions or commands of a program.
SANYO	A worldwide company that through engineering expertise has brought you this computer series.
SANYO-BASIC	A high level user friendly interactive language developed by SANYO.
SB-RUN	SANYO-BASIC with "engineering" functions.....to leave a large user memory.
SCAN LINE	The sweep of a cathode beam across the face of a cathode-ray tube or CRT, or other video display.
SCHEMATIC	An engineering diagram which depicts the interconnections between electronic devices.
SCROLL	To move text on a display upwards making room for text at the bottom of the display.
SECTOR	A portion of a track of data on a floppy diskette.
SOFTWARE	Programs, files, and data executable on the computer.
STACK	A reserved part of memory used to store temporary

information.

STAT — CP/M acronym for a directory list command. See Chapter 6.

SUBROUTINE — A much-repeated section of a program which can be accessed by "calling".

SYNTAX — The rules for instruction expression in a given language. A mistake in entering an instruction may result in a "SYNTAX ERROR".

SYSTEM DISK — A special diskette usually provided with the system that contains the operating system, languages, utilities, monitors, diagnostics, and other software for system use and initialization.

TEXT — Symbols entered for and by a program.

TRACK — One "band" of data on a floppy diskette.

VARIABLE — A location in memory where data can be placed. Also refers to the actual data itself.

VECTOR — A memory or program pointer for a "jump-to" location. Vectors are used to re-direct processor activity.

WARM-START — A restart of computer operations after an abortion of operating system control.

CHAPTER 5 - PROGRAMMING AND FLOW RELATIONSHIPS, OPERATING SYSTEMS

Introduction

Your SANYO series computer will respond to a program either created by you or entered from the storage files. A program defines a series of operations for your system to process. In this chapter we will focus on the basic parameters that make up all programs and how the instructions are processed. A knowledge of the basic techniques will help you in your application programming.

Instructions and Form

We all can take instructions. Your SANYO product does so as well in the form of a series of commands from a program. The program exists in the form of a list of numbered commands. The commands are in sequence.

Each command must be understood by the computer and so must be presented in the proper format or what we call correct "SYNTAX". If you present a command to the machine that is improper in its syntax you will get an error. The correct syntax for each command will be discussed in Chapters 8 and 9. We must always be aware of the abilities of the computer to process only known commands.

The SANYO series of computers can understand "enhanced" syntax with some ease. Here we refer to those commands that have spaces added to them or other commented symbols to make the meaning of the command more obvious to the user. A full range of possibilities has been preprogrammed into the syntax "look-up" list to give you a wide range of variable formats. An interesting feature of SANYO computers is their ability to list your programs in a "re-done" form to show you how it has interpreted the meaning of your commands.

Instructions take the following forms.....

DECLARATIVES	These set a constant or variable in the body of the program and in general declare a fact to the computer. Example: PHI = 3.1417
DEFINITIVE	These define to the computer a relationship that it will use over and over again. For example: DISPLAY DEVICE = CRT
DIRECTIVES	These are directly executable commands which are done immediately. For example; PRINT"HELLO WORLD" or STOP
COMMENTS	These are non-executable remarks for your use only in annotating the program. The REMark statement is such a comment.
TRANSFERS	These are instructions which transfer the program flow to another part of the program. For example: GO TO 100

FUNCTIONS — These define to the computer a relationship that you wish it to know. For example:
A = 3.14 + 7 + B + C

The Operating System

A computer is a memory based machine that, upon powerup, has nothing in its memory banks. Your SANYO computer has over 65,000 locations of memory that it can use to store various forms of information. A special memory apart from this large memory bank is used to pre-load the "operating system" into the computer. This smaller special memory is called a Read-Only-Memory or ROM and contains a special loader program called a "Bootstrap Routine" that will load in from the diskette system the required programs for system initialization.

Consider the "operating system" as a series of programs that will perform the "house-work" chores for the computer. The operating system programs also have the intelligence to be able to dialogue with you, the user....accepting commands and executing them.

The operating system boot will do the following upon power-up.....

1. Clear out the memory space available and test it.

2. Load in the Operating System.

3. Initialize the system by defining what input and output devices will perform what chores.

4. Establish diskette communications through a port structure.

5. Clear the screen and prompt you with a ready status.

The SANYO series of computers use a system called **CP/M** as their standard operating system. This conversational system is a world-wide standard for communication between man and machine. Chapter 6 covers CP/M in detail. Under the CP/M structure other operating systems can be called. **SANYO-BASIC** is an example of yet another operating system. More about this later.

Flow

Your computer will process instructions one at a time. The internal speed is of an order of magnitude in the millionths of a second per instruction. The greatest advantage computers have over the abilities of man is speed. Even with phenomenal speed, however, the computer takes time to interpret the instructions. This time it needs to decipher what you wish it to do.

Let's look at the structure of the processes the computer must go through to interpret and execute your commands in a meaningful way.....

You enter instructions into the computer in a USER LANGUAGE

These instructions are broken apart character

by character by the Operating System and checked for correct syntax. As it does so it starts to build a TOKEN LANGUAGE program. This is all done by the INTERPRETER of the Operating System. The machine then.....

Takes the TOKEN LANGUAGE program and prepares it for execution...

The TOKEN LANGUAGE program is written in a special form of MACHINE CODE that will process from the start to the end of the TOKEN LANGUAGE program. The Operating Sytem will then.....

EXECUTE the Program (using the TOKEN LANGUAGE) and will then...

RETURN to the Operating System for more commands.
If you enter more commands or edit your program the Operating System will automatically correct the Token Language Listing to reflect your desires. You have an allocated number of "free bytes" or locations into which to enter your commands for interpretation.

OPERATING SYSTEMS, then, in summary:

1. Provide interaction between man and machine.

2. Perform "house-keeping" chores within the memory.

3. Allocate input and output parameters.

4. Establish Tokenized Programs for Execution and monitor them.

5. Interpret your commands.

6. Are called from the diskette system by a "boot-strap" loader.

Summary

You will be using an Operating System to write programs on your SANYO series computer. As such we have seen that you will....

1. Enter commands in a form or syntax that is understood by the resident INTERPRETER of the Operating System.

2. Enter commands in a sequence that makes sense and flows.

3. After having entered the commands you will execute or RUN your program to achieve the desired results or you will LOAD from a diskette file programs written by others that will RUN under their own sequences.

4. Wishing to SAVE your programs you will output them to the diskette storage file for further use.
5. Understand that the abilities of the computer are only limited by its understanding of the task (commands) you give it to

process.

CHAPTER 6 - INTRODUCTION TO CP/M-86

Introduction

This chapter will introduce you to the operating system called CP/M-86 which has become a standard in the industry. Your Sanyo series computer implements this system in a most useful way. Here we will review the basic features of this most desirable of structures.

CP/M-86 stands for Control Program for Microprocessors. This program resides on the diskette furnished with your system. When your SANYO first "boots" up, the CP/M-86 file is automatically read and processed into your computer's memory.

Once loaded, CP/M-86 will allow a variety of processes tooccur in the machine. For example, it will send text to the printer, read and process information from the keyboard, display information on the screen and allow data low between peripheral devices. In addition, CP/M-86 will allow internal chores to be performed invisible to the operator. These "house-keeping" tasks include the management of disk space and the allocation of memory in the computer.

CP/M-86 is a tool which allows a dialogue to occur between the operator and the computer that is application oriented. By inputting general commands the many facets of CP/M-86 will can be used.

CP/M-86 usually comes with other types of programs merged with it. These sub-programs are designated as utilities and are considered the features of the operating system. Such system calls as PIP and STAT are in reality seperate programs under the CP/M-86 umbrella. We will cover these in detail.

Bringing Up CP/M-86

The SANYO 4000 series of computers all bring up CP/M-86 automatically upon system power-up. Make sure that the system disk shipped with your computeris placed in drive A (if you have single drive, the drive on your console).

When the power on sequence is started you will hear a series of noises as the disk file is automatically read, the screen will display a message followed with what we call the CP/M-86 PROMPT which is always:

"A>"

This means both you and the computer are ready for business. The A> means the operating system is waiting for you to specify a file on diskette A for the operating system to fetch or for a directive command within the system use parameters.

You have just performed a "cold start" or power-up sequence. This is different from a "warm start" sequence or initialization. To perform a warm start sequence simply do a CONTROL C sequence by depressing first the CONTROL key and (with the CONTROL key down) depress the C key.

Help with CP/M-86

The HELP command on the operating disk will provide descriptions and examples for the following topics:

ASM86	COMMANDS	COPYDISK	DDT86	DIR	DIRS	ED	ERA	FILENAME
GENCMD	REN	STAT	SUBMIT	TOD	HELP	TYPE	PIP	USER

Enter HELP at the A> prompt. The following message will appear:

```
HELP UTILITY VX.X
AT "HELP>" enter topic(,subtopic)...
```

Subtopics are listed in the information display(i.e. EXAMPLES).

Working with Drive B

After a prompt message you can immediately switch over to the second disk file by typing a B: after the A> prompt:

"A>" B:(CR) Note:(CR) inicates Carriage Return

The operating system will immediately respond with:

"B>"

You are now ready to read system files from disk file B.

To return to diskette file A an A: must be typed.

Typing in the File Name or Command

If you type in a file to be read after the prompt character is displayed and the file is non-existant or the command is unknown, the CP/M-86 operating system will repeat your typed in text followed by a question mark(?):

```
A> Nofile (CR)

Nofile?
A>
```

Correction of Entry

If your entry was incorrectly typed you may correct it by doing one of two things. The first is to type a CTRL-X sequence. This will make all text to the left of the prompt vanish....leaving the cursor in the proper position for re-entry. The second way is to simply allow the error to appear in the prompt field and depress the RETURN Key. This will cause a File Not Found status and will abort with a re-prompt message.

Examination of the Directory

Each disk file has on its corresponding diskette a file that is the directory for that particular diskette. Under CP/M-86 you may see this list of available programs and files by simply typing DIR followed by a carriage return after the CP/M-86 prompt.

Example:

```
A> DIR (CR)

A:PIP          CMD
A:ASM86        CMD
A:STAT         CMD
A:HELP         CMD
A:BASIC        CMD
```

If your display goes by too fast press the CTRL and S keys at the same time. This will stop the display. Repeat this and the display will continue to pass by again.

In the example above, the file names appear at the left and the type of file is on the right. Files are basically either COMMAND or CMD files, BASIC or BAS files, INTERMEDIATE or INT files.

One must realize the various file differences as BASIC files cannot be easily called from the CP/M-86 operating system and must be called from a BASIC operating system. The BASIC operating system can be called from the directory under CP/M-86 as it is a CMD file on the diskette. Once BASIC is called, a different system call scheme and hence operating system is in effect.

To call for the file on a specific drive follow the DIR command with the drive number.....A> DIR B: (CR) for example requests the B diskette files.

Under CP/M-86 1.1, the version for the SANYO 16 bit series, the directory may appear in the form of four columns. This is an easy way to avoid many line feed operations on the screen.

DIRS

DIRS displays the names of system(SYS) files in the directory of on-line disk whereas DIR displays the names of non-system files.

File Names

The files from the directory, or your own files for that matter, are labeled in a way that provides for a major name and an "extension". The extension is a group of three characters placed after a period. This extension can be anything that makes sense to you as an operator. For example:

```
TEXT.RON
```

This might refer to a text file that belongs to RON.

The file name (here TEXT) can be up to 8 characters long.
Filenames cannot contain the following symbols.....

> < . , ; = * ?] [:

Extensions are fairly universal,but there are some that are reserved. The following list of extensions should not be used except as defined:

EXTENSION	TYPE
CMD	Used in a Command File
ASM86	Required for assembly source files
LST	Required for assembly listings
H86	For hex files
BAS	BASIC files
INT	Intermediate files in BASIC
BAK	Editor created back-up files
$$$	Temporary Scratch files
SUB	Text file for batch submits

One can use a question mark "?" in the file name description to indicate a"don't care condition" for file match. In this case the letter position will be ignored in the actual matching search. For example:

TEX?.?EF will match to TEXT.DEF, TEXS.GEF, TEXP.XEF and others.

Use of the Type Command

The TYPE command can be used to indicate and display on the screen any ASCII file. This is a very efficient way to examine text files. The format for the command is as follows, typed after the prompt:

TYPE d: xxxxxx.yyy

Where d is the drive number (A or B)
Where xxxxxx.yyy is the file name

If you wish the file to be printed depress a CTRL-P before giving the command. Of course if the printer option is used the text display is greatly slowed down.

It is wise to use the TYPE command at those times when a quick examination of text is necessary, such as directly after the creation of a new file.

Use of the REN (RENAME) Command

Any file name can be changed to be a new name. The REN command permits this. The format for the command is as follows;

REN newname = oldname

A file disk number may be designated but the file names must be on the

same disk drive number.....the example below is valid:

```
A> REN B:NEWFILE.TXT=OLDFILE.TXT
```

Use of the ERA (ERASE) Command

Many times it is necessary to delete a file name from the diskette directory. To do so requires the use of the CP/M-86 ERA or ERASE command. The format of the command is simply:

```
ERA d:xxxxx.yyy

Where d is the disk number (A or B)
Where xxxxx.yyy is the file name
```

Or with the prompt message as an example.....

```
A> ERA NONEED.TXT
```
or....
```
A> ERA B:NONEED.TXT
```

Please keep the following in mind:

ERA IS A COMMAND THAT CAN BE DANGEROUS! USE IT CAREFULLY!!

Printer Interaction

If you have a line printer attached to your system you may send your chosen file to it by a very simple technique. Depress CTRL and P at the same time. Now anything typed on the screen will appear on the printer. Use of the TYPE command described earlier will dump the file both on the screen and onto the printer. You can turn the printer option off by repeating the CTRL-P sequence. Note how much slower it takes to TYPE process when the printer is on and selected.

Use of PIP

There is a very powerful command called PIP which will allow you to do a large variety of things in CP/M-86. The primary use of PIP is to copy or to transfer files. PIP works between two devices. PIP will transfer the files and at the same time it can process or redo them during the transfer.

COPYING FILES

The following is a way to copy a file (here called ORIG.TXT) onto the same diskette as the source and to call the copy COPY.TXT.....

```
A> PIP A:COPY.TXT=ORIG.TXT   (CR)

A>
```

In the above example the PIP command was executed from the prompt. PIP may also be presented as a "program" by entering into the program mode of PIP, designated by a system prompt with the character "*". To

do so type PIP after the prompt and watch the system respond with a "*".....

```
A> PIP  (CR)

* A:COPY.TXT=ORIG.TXT  (CR)

* B:=A:COPY.TXT   (CR)

*    (CR)

A>
```

Here we have done the following:

1. In the first statement we have entered the "program mode".
2. Next we have made a copy onto A of an existing file called ORIG.TXT, calling the copy COPY.TXT. The original file was on A.
3. We now make a copy onto disk B of the file called COPY.TXT on A.
4. By typing a return only we exit back to the CP/M-86 prompt.

There is a special character "*", which when specified in a PIP command will cause all letters in its position on corresponding files to be automatically matched. It is best to use an example. Suppose you wished to copy all CMD type files from one diskette to the other. You may do so with a single command:

```
A> PIP B:= *.CMD     (CR)
```

Here the "*" tells us that only the .CMD extension will be used for the matching and all files that end in .CMD will be transferred from drive A to drive B.

COPYING ALL FILES

Suppose you wished to copy all the files from one drive to the other. The powerful PIP command and the "*" option allow you to do this very easily. The format for the command would be.....

```
A> PIP B:=A:*.*  (CR)
```

Here all files on drive A will be copied onto Drive B

NOTE: One can copy files from diskette to diskette, not systems. This means CP/M-86 on a disk will not be copied. There is a special program to copy system files.....LDCOPY, explained elsewhere.

MERGING FILES TOGETHER

The merging of files together is called concatenation. The PIP command allows one to very easily merge files by their name.....

```
A> PIP  (CR)

* MERGED.TXT=TEXT1.TXT,TEXT2.TXT,B:TEXT3.TXT
```

Here a new file called MERGED.TXT was created from three text files of which two were on drive A and the other on drive B.

NOTE: We have transferred TEXT files (indicated by .TXT). PIP can be used to transfer hex and non-text files as well under special conditions.
Refer to the literature list at the end of this chapter for further information.

COPY TO OTHER DEVICES

A file can be sent to a number of different devices. A printer, a paper tape punch, a cassette recorder, a magnetic card unit or devices. Likewise a card or tape reader, a disk or other peripheral can be the source of the data. PIP will enable us to transfer data between such devices

Certain devices are allowed as the logical device name for PIP to recognize:

CON:	Refers to the console or terminal: keyboard/display
AXI:	Refers to the paper tape or card reader (input only)
AXO:	Refers to the paper tape or card punch (output only)
LST:	Refers to the listing device usually a line printer

Certain devices are allowed as the physical device type for PIP commands-

TTY:	Refers to a console or terminal
CRT:	Refers to the console list device (screen)
PTR:	Refers to the paper tape reader (input)
PTP:	Refers to the paper tape punch (output)
LPT:	Refers to the list printer
UC1:	Refers to the user defined console device
UR1:	Refers to the user defined reader device
UP1:	Refers to the user defined punch device

UL1: Refers to the user defined list device

To use the file names one simply defines the destination and the source within the desired PIP statement. For example, suppose we wished to transfer the contents of a file called MENU.TXT to the line printer device designated by the device type LST. Assume the file is on disk B.

```
A> PIP  (CR)

* `LST:=B:MENU.TXT    (CR)

*   (CR)

A>
```

SPECIAL PARAMETERS DURING COPY AND TRANSFER

PIP has the ability to condition data as the transfer occurs. To do so a special set of parameter codes has been defined. This set is enclosed in a set of braces within the command. Each parameter can be specified uniquely or in a composite grouping. The format is shown here in example form:

```
A> PIP B: xxxxx.yyy = zzzzz.ttt [ abcdef...n ]
```

Here a b c d e and so on are parameters that will be executed during the transfer process.

The following table will show what some of these parameters are:

E	Echo (redisplay) all copy on the terminal screen
L	Translate all upper case characters to Lower case
U	Translate all lower case characters to Upper case
Dn	Delete characters past coulumn "n" on a printer
B	Read until you see a "X-off" character in ASCII
H	Transfer using the Intel hexadecimal format
Pn	Page ejects every "n"th line
V	Verify the transfer process with a re-read

There are others defined in the mentioned literature.

Example: * LST: = BOOK.TXT[P60LE]

Here a file called BOOK.TXT is transferred to the line printer with page ejects every 60 lines as indicated by P60, all upper case is translated to lower case with the L format, and the text is simultaneously listed on the terminal with the E format. Note commas are not used in the field.

Use of LDCOPY

One may wish to make a copy of the system diskette. PIP will not do this as the PIP command copies only files, not systems. Further, the first two tracks of a diskette hold special formats and data that are difficult to copy. For this reason we have in CP/M-86 a command called LDCOPY which will allow us to copy the system diskette.

The format of the command is .....

A> LDCOPY (CR)

LDCOPY VERS 1.0 256 byte/sec 1982.9

Source Drive Name A

Source onA Then Type Return

At this point make sure the source system disk is in drive A.

Function Complete (Here the read has occured)

Destination Drive Name (or Return to Reboot)B

Destination onB Then Type Return (CR)

Make sure your new disk is on drive B

Function Complete

Destination Drive Name (or Return to Reboot)

Depress (CR) to terminate LDCOPY

Use of the STAT Command

There is a way to display the status information for the peripheral devices. This is called the CP/M-86 STAT command. It can also show us the available disk space allocations for a particular drive.

The simple STAT command call displays the sizes of remaining disk space and the file allocations. The format is:

```
A> STAT    (CR)

A:R/W, SPACE: 160K

A>
```

Here we are told that diskette A has 160k bytes free for use and that we can both read and write to this diskette. If the message returned was R/O we would know we can read only as the write mode has been disabled.

Let's now turn our attention to other devices.....
By typing the following sequence you may see the DEVICE ASSIGNMENTS in your SANYO computer.....

```
A> STAT DEV:    (CR)

CON:=CRT:

RDR:=TTY:

PUN:=TTY:

LST:=CRT:
```

These show us the various devices assigned to each of the peripheral files. To change the device simply do this:

```
STAT lll: = ddd:   where lll: is the logical file
                   where ddd: is the device type
```

Lets now look at the status of a diskette.

```
A> STAT B:    (CR)

BYTES REMAINING ON B:  220K

B: R/O

A>
```

Here of course we know that drive B has 220K bytes free and that it is a read only (write protected)

drive.

IF you attempt to write to the file. You will get a "BDOS ERR ON B: READ ONLY".
You can change file status with a status statement such as:

```
A> STAT B: = R/O    (CR)
```

Here we have made the B drive a read only drive.

STAT can be used to extract the file status of files from diskettes. To do so merely type in the STAT command as shown:

```
A> STAT B: WHAT.TXT    (CR)

RECS   BYTS    EXT   D:FILENAME.TYP

10     2K      1     B:WHAT.TXT
```

Here we see that the file called WHAT.TXT is a 2k byte files occupying 10 records on the diskette. One"extent" or free allocation is available in the file.

STAT can be used to display the status of ALL the files with the following command....

```
A> STAT B:*.*          (CR)
```

See "Copying All Files" section for more details

Other STAT commands are STAT X: DSK: and STAT X: USR: where X: is the drive name.

STAT X: DSK: displays the configuration of the selected disk drive. It is self explanatory.

STAT X: USR: displays the information of who the active user and active file are and how many users there are on the system.

Use of the ED Command

CP/M-86 contains a full editing program called ED. You may access ED, which is on a CMD file by doing the following:

A> ED xxxx.yyy

Where xxxx.yyy is a TXT file

This call creates a buffer or allocation in memory that will allow your file called to be edited. It also deletes any BAK or backup file descriptor that matches your chosen file name. The ED mode is terminated with an E command at which time the ED protocol updates the source file and creates a backup file of the original source file.

It is not the intent of this manual to cover the ED command in great detail, please refer to the literature suggested for more infomation. We can, however, give you the following chart of commands which can be of some assistance.

COMMANDS	MEANINGS
nA	Append Text n lines from source file to the edit file.
+nD -nD	Delete n characters forward (+) or backward (-).
E	End the EDIT mode.
Fstring	Find the "string" specified
I	Insert a new line of text after the character pointer.
+nK -nK	Kill n lines forward (+) or backward (-) from character pointer.
+nP -nP	Print n number of pages forward (+) or backward (-).
Q	Quit
V	Turn on line number display
OV	Display free bytes left
nW	Write out with ".$$$" extension n lines to temporary output file.

Use of the ASM86 Command

This CP/M-86 call is used by assembly language programmers. The call to ASM86 from CP/M-86 turns an assembly language source file into a machine code file of the .HEX type.

The format for the command is:

```
A> ASM xxxx.yyy     (CR)

Where xxxx.yyy is a source file
```

ASM86 will translate (assemble) the assembly language source file into INTEL hexadecimal notation as binary code.

Use of the DDUMP Command

The DDUMP call to CP/M-86 loads onto the screen display device any file on the diskette in a special hexadecimal format.
The format for the command is:

```
A> DDUMP      (CR)
```

The following message will be displayed:

```
DDUMP - DISK DMP - VER X.X  (DATE)
*      (The * indicate DDUMP command mode)
```

Key in the following in the command mode:

```
*[drive name,]tracknumber[,[sector number
 ,EDIT]]   (CR)
```

Example: Dump the comtents of sector 1 on tract 1 of drive b.

```
*B:,1,1   (CR)
```

Data can be changed using the edit mode:

```
*B:,1,1 EDIT
```

When the - is displayed, key in the address where the new data will be written. Replace data one byte at a time.

Return from edit mode by entering a **W** to write replacement data on the disk and return to command mode, or by entering a **Q** to return without writing replacement. Refer to Chapter 13 for detailed examples.

Use of the DDT86 Command

The CP/M-86 structure has a monitor built into it accessible under the DDT86 call. This program loads DDT86 into the temporary storage memory and awaits the loading of another program or file upon which it

operates. DDT86 has its own commands for the insertion of data values, displaying memory locations, saving comments, setting breakpoints and the like. It is usually used for debugging purposes.
To call the DDT86 under CP/M-86 type:

```
A> DDT xxxx.yyy    (CR)

The xxxx.yyy file is either a
.HEX or a .CMD type
```

The I command inputs a .HEX or .CMD file into the computer.

The R command reads the file and presents a "load message" that consists of the NEXT PC (program counter) status. The NEXT address is followed by the number of page.

The GO command terminates the DDT mode.

Use of the SUBMIT Command

The command SUBMIT calls a program that accepts the file file name .SUB and builds a file $$$.SUB which is executed after a warm start. The $$$.SUB file's command lines are executed until the file is exhausted.

A file is created using the ED command that is a .SUB file. These files have arguments expressed as variables designated $1, $2, $3 and so on.

The format of the command allows variables v1, v2, v3 and so on to follow the file name. These are substituted for the $1, $2, $3 variables on a one-to-one basis. Command format:

```
A> SUBMIT v1 v2 v3.....
```

Specifying Files with Commands

File names in CP/M-86 (DOS) are specified in the following form:

mmmmmmmm. nnn

where m and n are characters. A file name may contain up to eight (8) characters followed by a period and three (3) characters representing a qualifier. Qualifiers do not have to be specified.

Any character except < > : ; = . ? : , [ ] may be used for a file name.

Sample File Names

TEST

TEST-1. BAS

12345678.ABC

MBC1000.123

SIBOOM. BAA

!"#$%&'(.ABC

File Referencing

Files may only be referenced if they have a unique name. If you assign two files the same name, the second file will write over the first file and you will loose the data on that file.

File names may not contain any command names.

Several files may be referenced at once through the use of the question mark ? or the asterisk *.
Each question mark represents any single character.
Examples:

ENTER	DEPRESS	RESULT
DIR ^ BA??????.???	RETURN	All files on the disk that begin with BA will be displayed.
STAT ^ MATH?.BAS	RETURN	Calculates the total area occupied by the files that have a five character name that begin with MATH and have the qualifier .BAS on the current disk. The value is displayed.
PIP ^ B:=CHAPTER?.TXT	RETURN	Copies all the files that have CHAPTER as the first seven characters and an unknown eighth character with the qualifer TXT to the disk on drive B:.

Asterisks represent an entire file name or qualifier.

Examples:

ENTER	DEPRESS	RESULT
DIR ^ *.*	RETURN	All files on the curent disk are displayed.
STAT ^ A*.*	RETURN	Calculates the total area occupied by all the files beginning with A (with up to seven more characters) with any qualifier and displays the value.
PIP ^ B:=*.CMD	RETURN	Copies all the files with CMD format (qualifier) to the disk in drive B.

. is equivalent to ????????.???

The question mark and asterisk may be used together.

ENTRY	DEPRESS	RESULT
DIR ^ AB???.*	RETURN	Displays the names of all five character name files on the current disk with the first two letters AB and any qualifier.
STAT ^ *.CO?	RETURN	Calculates the total area occupied by all files with a qualifier that has the first two letters CO.
PIP ^ B:=*.???	RETURN	Copies to drive B all files on the current drive. This is equivalent to *.*

Drive Specification

The drive may be specified before the name of the file. If the system has only one drive, only this drive may be specified.

ENTER	DEPRESS	RESULT
DIR ^ A:BASIC.CMD	RETURN	Displays BASIC.CMD from drive A.
DIR ^ B: *.*	RETURN	All files on drive B are displayed.
DIR ^ A: DATA*.*	RETURN	All files from drive A beginning with DATA (and up to an additional four more characters) with any qualifier are displayed.

If no drive is specified, the current diskette will be assumed.

If A> is displayed, the current diskette must be on drive A. If B> is displayed, the current diskette must be on drive B.

Change Drives

Enter the drive name followed by a colon (:) after the input symbol >.

ENTER	DEPRESS	RESULT
B:	RETURN	B> is displayed indicating the current drive is now B.

Number of Files on a Diskette

A diskette may contain up to 128 files. If a diskette has a large unoccupied area even after storing 128 files, no more files can be stored on it. Files must be erased to make room for new ones. In DOS mode, the ERA command erases files.

Example:

ENTER	DEPRESS	RESULT

ERA TEST-1.BAS	RETURN	The file, TEST-1.BAS is erased

In BASIC, the KILL command erases files.
Example:

ENTER	DEPRESS	RESULT
KILL "TEST-1.BAS"	RETURN	The file, TEST-1.BAS is erased

Files should be copied using the FILECOPY Utility or the PIP Utility before they are erased if they are important.

Each time the contents of a file exceed 64 bytes, extensions are added to it. Extensions are set in 64 bytes units, each treated as a separate file. A file not exceeding 64 bytes is trated as having one extention only. A file exceeding 64 bytes, but not 128 bytes is considered to have two extentions. Depending on its size, therefore, a file may be treated as two or more physical files. You may have 128 files on the disk even though you do not have 128 individual files.
Use the STAT Command to know the contents of an extention (EXT).

ENTER	DEPRESS	RESULT
STAT^TEST-1.BAS	RETURN	RECS BYTES EXT ACC # # # # Bytes Remaining on X: #K

COMMANDS

Some commands may change, delete or destroy disk contents. Please read this section before you use any of these commands. Throughout this section, this symbol, ^, will mean that you will enter a space by depressing the space bar. The following resident commands are listed in alphabetical order.

DIR

Purpose

This command displays the name of specified files contained in a specified drive.

Description

1) DIR displays the names of all the file(s) on the current disk.

2) If the required file(s) cannot be found, the error message NO FILE will appear.

3) The DIR command does not display system files (files set by issuing STAT). Use the STAT command to display system files.

Examples

ENTER	DEPRESS	RESULT
DIR	RETURN	All files on the curent disk are displayed.
DIR^*.*	RETURN	All files on the current disk are displayed.
DIR^AB??.*	RETURN	All four character files beginning with AB and having any qualifier on the current disk are displayed.
DIR^B: AB??.*	RETURN	All four character files beginning with AB and having any qualifier on drive B are displayed.
DIR^TEST-1.BAS	RETURN	File TEST-1.BAS on the current disk is displayed.
DIR^*.BAS	RETURN	All the files on the current disk with the qualifier BAS are displayed.

ERA

Purpose
This command erases file(s) from a diskette.

Description

1) This command may be used to make room for additional files on a diskette.

2) If a diskette contains important files, remove then to a read-only (R/O) file by issuing the STAT command.

3) If the ERA^** command is issued, the prompt ALL (Y/N)? will be displayed. The files will not be erased until the operator enters a Y in response.

Examples

ENTER	DEPRESS	RESULT
ERA^TEST-1.BAS	RETURN	TEST-1.BAS IS erased
ERA^**	RETURN	ALL(Y/N)?
ERA^*.BAS	RETURN	All files with the BAS qualifier on the current disk are erased.
ERA^AB??.*	RETURN	All four character files beginning with AB and having any qualifier on the current disk will be erased.
ERA^B:*.BAS	RETURN	All files with the BAS qualifier on drive B are erased.

FORMAT

Purpose
Initializes a blank disk.

Description

1) Newly purchased diskettes cannot be used until they are initialized by the FORMAT command. Initializing organizes the disk so that it maybe written to and read by the computer.

2) All diskettes supplied wth the computer have been initialized.

3) Issuing the FORMAT command for a diskette containing programs and data erases all these contents.

4) If the system diskette is left in drive A: after entering the FORMAT command, all contents of the diskette will be erased. The diskette to be initialized must be placed in drive A before an F is entered to start formatting.

Example

ENTER	DEPRESS	RESULT
FORMAT	RETURN	Mini-floppy formatting program (c) 1982 with 2 side, double density,256 bytes/sectors,16 sector/track, 80 track/side. disk on A: drive, then type "F" (or ^C to reboot)(or "T" to test only)

The prompt indicates that you must use a double sided, double density, soft sectored disk. Insert the disk to be formatted on drive A, then enter an F only (no carriage return) to start formating. If you do not wish to format, depress the CTRL (control) Key and key C at the same time. This will take to back to the DOS mode. You will see the input symbol A>.

If you enter a T, the disk will be tested to see if it has been formatted and if it was formmated correctly.

Formatting is completed when the following prompt is displayed:

diskette test program go ahead!

Do not remove the diskette. The system is now testing the diskette to ensure correct formatting. This will take about two minutes. Testing is completed when a beep sounds and the following prompt is displayed:

certified 80 track!

You may now format another disk or depress the Control Key and C Key to exit to system level when the "Mini-floppy formatting..."prompt is again displayed.

REN

Purpose

This command is used to change the name of a file.

Description

1) The prompt, FILE EXISTS, will be displayed if a file already occurs with the same name. The old file name will not be changed.

2) Drive names may be specified. If only one drive is specified, it will be assumed that both files are on that drive. The drive names must be the same or the name will not be changed.

3) If the old file name cannot be found, the prompt, NO FILE, will be displayed.

Examples

ENTER	DEPRESS	RESULT
REN^TEST3.BAS=TEST2.BAS	RETURN	TEST2.BAS is renamed TEST3.BAS
REN^B:DEF.CMD=ABC.CMD	RETURN	ABC.CMD on drive B is renamed DEF.CMD

STAT

Purpose

Displays the status of diskettes and the I/O unites. It also changes device assignments.

Description

1)The STAT command may be used to display the following:

.File size
.Sizeof empty or read-only area on the disk
.Disk use

or to change the following:

.Software based read-only setting
.System file setting

A. File Size

File size is expressed as follows:

Recs	The number of data records written on the file. One record = 128 bytes.
Bytes	The quantity (in KBytes) of data written in the file.
Ext	The area used by the files on the diskette, expressed in units of extentions.
Size	The number of data records written in the file. For a sequential file, this is the same as the Recs value. In a random file, where the records are not sequentially aranged, the value is estimated and may differ from the actual number of records.
Acc	Indicates whether the file is Read-Only or Read-Write.
X:mmmmmmmm.nnn	Specifies the diskette drive (X) and filenames (mmmmmmmm.nnn). If the file name is enclosed within parentheses, the file must be a system file.

Example

ENTER	DEPRESS	RESULT				
STAT^TEST-1.BAS^$	RETURN	Size	Recs	Bytes	Ext	Acc
		ssss	rrr	bbbk	ee	R/O X:mmmmmmmm.nnn

B. Empty and Read-Only Areas on a Diskette

The STAT command may be used to find out the usable area on a diskette.

Example

ENTER	DEPRESS	RESULT
STAT	RETURN	X: R/W, Space: nnnk

X: represents the drive name and nnk the usable area (k for KBytes). R/W indicates that the diskette is Read-Write. The usable area for all the disks inserted in all drives connected to the computer will be displayed.

STAT^X:	RETURN	Bytes Remaining on X: nnnk The following also may appear: X: R/O, Space: nnk This indicates that the diskette is specified for Read-Only by the software (in response to STAT).

C. Diskette Use

The state of a specified diskette on a current drive may be displayed.

Example

ENTER	DEPRESS	RESULT
STAT^A:DSK:	RETURN	A: Drive characteristics 4992: 128 Byte Record Capacity 624: Kilobyte Drive Capacity 128: 32 Byte Directory Entries 128: Checked Directory Entries 512: Records/Extent 32: Record/Block 32: Sectors/Track 4: Reserved Tracks

D. Software Based Read-Only Specification

The STAT command may be used to specify a part (expressedin units of files) or the entire disk for Read Only.

Example

ENTER	DEPRESS	RESULT
STAT^A:=R/O	RETURN	Restricts all files on drive A to Read-Only.
STAT^TEST-1.BAS^$R/O	RETURN	Restricts file TEST-1.BAS to Read-Only.
STAT^TEST-1.BAS^$R/W	RETURN	Returns file TEST-1.BAS to Read-Write.

E. System File Setting

A file may be made a system files by using the PIP or the DIR command to prevent others from knowing about it. This however, is not a full protection, because a user may know its existence and size by using STAT followed by the file name.

Example

ENTER	DEPRESS	RESULT
STAT^TEST-1.BAS^$SYS	RETURN	File TEST-1.BAS becomes a system file.
STAT^TEST-1.BAS^$DIR	RETURN	File TEST-1.BAS is released and no longer a system file.

2) In CP/M-86, two device specifications are possible, an actual device name and a device name defined in DOS.

The logical device names are:

CON: system console

AXI: input-only device

AXO: output-only device

LST: list device

The current physical device names are:

CRT: display screen

PTR: paper tape reader

PTP: paper tape punch

LPT: line printer

The state of assignment may be displayed.

Example

ENTER	DEPRESS	RESULT
STAT^DEV:	RETURN	CON : is CRT:
		RDR : is PTR:
		PUN : is PTP:
		LST : is LPT:

Assignments may be changed.

Example

ENTER	DEPRESS	RESULT
STAT^CON:=TTY:	RETURN	The actual device name is changed to TTY: (terminal).
STAT^RDR:=TTY:,LST:=CRT:	RETURN	The input-only device is changed to TTY: (terminal),and the list device is changed to CRT:(screen).

Note in this last example that two or more device names may be defined in one STAT command. Each must be separated by a comma (,).

SUBMIT

Purpose

The SUBMIT command is used to execute batch processing.

Description

1) The SUBMIT command retrieves the specified file and carries out batch processing according to the commands cataloged in the file and the given parameters. For this,it genrates the $$$.SUB file on the current disk. CCP processes the commands in the orde in which they are generated on the $$$.SUB file.

2) The SUB file must be entered in advance on the diskette in drive A to use the SUBMIT command for it.

3) If an error occurs during the execution, CCP stops the execution immediately and erases the contents of file $$$.SUB.

4) During execution, parameters P1 - P9 correspond to $1 - $9 written with the commands in the SUB file. $n is replaced by Pn. $0 indicates that the file specified by SUBMIT must be treated as parameter P0. See example 1.

5) To include $ as part of a command in the SUBMIT file, specify $$. See example 2.

6) A SUBMIT file may be called during execution of the SUBMIT command entered in another file.

7) The character string contained by a command in a SUBMIT file can run up to 128 bytes. The total length of $$$>SUB, including the parameters generated, is 2048 bytes.

8) The SUBMIT file must end by depressing the RETURN Key, or LINE FEED Key.

9) The RETURN Key and LINE FEED Key are automatically included when a SUBMIT file is created by BASIC. To terminate BASIC, depress any key while the commands are being used.

Example 1

Assume that file DISK.SUB specified by SUBMIT contains the following commands:

```
DIR    $1

BASIC $2
```

ENTER	DEPRESS	RESULT
SUBMIT^DISK^A:^TEST:	RETURN	A: is converted to $1 and B: to TEST.

```
DIR    A:

BASIC "TEST"
```

These commands are stored in file $$$.SUB and CCP starts executing them in the order in which they appear.

The commands needed to create the DISK.SUB file are:

ENTER	DEPRESS	RESULT
BASIC	RETURN	BASIC-3 Ver X.X nnnn Bytes free
		Ready>
10 OPEN "O" #1, "A: DISK.SUB"	RETURN	
20 PRINT #1, "DIR $1"	RETURN	
30 PRINT "BASIC "+CHR$ (&H22)+"$2" +CHR$($H22)	RETURN	
40 CLOSE #1	RETURN	
50 END	RETURN	

Example 2

ENTER	DEPRESS	RESULT	
DIR^$$$.SUB, PRINT^ #1, "DIR^$$$$$$.SUB"	RETURN		Includes $ as part of a command in the SUBMIT file.

TYPE

Purpose

This command dsiplays the diskette file contents on the console.

Description

1) The display may be halted by depressing first the CTRL Key, then the S Key. Resume by the depressing the sae keys.
2) A display my be terminated and control returned to CCP by depressing first the CTRL Key, then the C Key. The display also may be terminated by depressing ay key except CTRL or S.

Example

ENTER	DEPRESS	RESULT
TYPE^TEST-1.BAS	RETURN	The contents of TEST-1.BAS will be displayed on the screen.

PIP

Purpose

Copies files and prints the content of a diskette.

Description

1) The following examples illustrate how PIP copies files within one diskette.

Examples

ENTER	DEPRESS	RESULT
PIP	RETURN	* Command lines may be entered without PIP preceding it (i.e. PIP A=C).
A=C	RETURN	Copies file C contents into file A.
TEST-2.BAS=TEST-1.BAS	RETURN	Copies file TEST-1.BAS contents into TEST-2.BAS.

3)The followng examples illustrate how files are copied from one diskette to another.

Examples

ENTER	DEPRESS	RESULT
PIP	RETURN	* Command lines may now be entered without PIP preceding it.
B:=A: TEST-3.BAS	RETURN	Copies file TEST-3.BAS on drive A to the diskette on drive B.
B:AAA=A:CCC	RETURN	Copies the contents of the CCC file in drive A into AAA file in drive B.
B:=A:*.CMD	RETURN	Copies all the files of format CMD in drive A into the diskette in drive B.
B:=A:*.*	RETURN	Copies all the files inserted in drive A into the diskette in drive B.

4) The following examples illustrate how files may be printed with the PIP command.

Examples

ENTER	DEPRESS	RESULT
CON:=TEST.BAS	RETURN	Outputs the file TEST.BAS in the console device specified by STAT. If the device is assigned as CON:=CRT:, then the result would be as if CRT:=TEST.BAS was entered.
LPT:=TEST.BAS	RETURN	Prints TEST.BAS at the line-printer.

5) Data in the files copied may be manipulated by specifying the required parameters within brackets after the file to be copied (i.e. PIP B:=TEST.BAS[E].

[ ]	**Explanation**
B	Specifies block mode transfer. The data is buffered till reception of the X-off characte (CTL S) from the source device. This function allows transfer of data from a sequential device to a disk file.
Dn*	Does not include, when copying the respective lines, the characters or digits from the nth character position to the end of the line. If n is specified as 0, 256 or more, or is omitted, then n=80 is assumed.
E	Displays all the characters transfered on the display.
F	Omits the form feed code in the line.
H	This specifies hexadecimal transfer. All data is checked to make sure that it conforms to the Intel Hex Format. Characters not required are eliminated during copying. If an eror occurs, a message appears on the console to prompt the operator to take adequate measures.
I	All ":00" records are ignored during transfer of data in the Intel Hex File Format(the H parameter is set automatically in response to the H parameter specification.
L	Converts upper case alphabetic characters into the corresponding lowercase characters
N	Adds a serial number to the beginning of each line, starting from one (1). If zero (0) appears at the begining of a line, it is deleted.

N2	Adds a serial number to the beginning of each line. An initial zero (0) is not suppressed, but a tab apears after the serial number in its place.
0	Transfers CMD formatted (other than ASCII) files. CTRL-Z (1AH) is ignored.
Pn*	Enters the form feed code after every n pages. If n is specified, as 0,1,256, or is omitted, n=60 is assumed.
Qs^z	Suspends copying data from the file or source device if the string "S" (CTL Z) is encountered.
Ss^z	Specifies start of copying after the string "S" is encountered. Both S and Q can be used to extract specific sections (like subroutines) from the file. NOTE: If a command is issued in the PIP format, strings, including S and Q parameters, are converted into uppercase characters. However, whenusing the auto PIP command (enter PIP depress RETURN) this conversion does not take place normally.
Tn*	Sets the tab code as n. If the user specifies 0 or a number above 256 for n, then n=8 is assumed.
U	Converts lowercase alphabetic characters into the corresponding upper case characters. This function is useful for terminals that cannot print lowercase characters.
V	Checks the data written by having it read again. Only a disk file may be specified as the destination.
Z	Resets the parity bit of the respective characters during transfer.

Examples

ENTER	DEPRESS	RESULT
PIP B:=FILE1.DAT[V]	RETURN	Copies FILE1.DAT to drive B and verifies.
PIP	RETURN	* Command lines may now be entered without PIP preceding it.
CRT:=TEST.BAS[L]	RETURN	The characters will be converted into the corresponding lowercase characters and output on the CRT.

6) The following additional devices may be provided for use with the PIP command:

NUL: outputs 40 null (do nothing) characters (00H) to the destination specified.

EOF: send end-of-file marker(CTL Z or 1AH) to the destination specified.

OUT: user custom destination device. PIP must be modified to use use this.

INP: user custom source device. PIP must be modified to use this.

PRN: same as LPT: device. The tabs stops are placed after every eight (8) columns and every line is numbered. The page feed character is inserted after every sixty (60) lines (as in NT8P).

7) File with a qualifier other than CMD are treated as ASCII files.

8) If a key is depressed during data transfer, the transfer operation will stop.

9) PIP is terminate with a Control Z.

Other CP/M-86 Data

We have attempted to present an overview of the major aspects of the CP/M-86 Operating System as implemented in your SANYO computer. There are many other functions in CP/M-86 which are not the subject of this introduction. We recommend for your further reading one of the many CP/M-86 texts available in the marketplace.

Informative books, both user and advanced, may be purchased directly from Digital Research. For more information about these books and manuals, write or call Digital at the following address:

Digital Research
P.O. Box 579
Pacific Grove, California 93950

408 - 649 - 3896

CHAPTER 7- PREFACE TO SANYO-BASIC PROGRAMMING

Introduction

Your SANYO computer uses as an available language SANYO-BASIC, a powerful but yet easy to understand set of computer instructions. With SANYO-BASIC you will be able to direct your SANYO's every action. SANYO-BASIC comes to you as a CMD file (BASIC CMD) on the System Disk. Various features have been added to bring to sharp focus the many features of your SANYO computer.

History

BASIC as a language was originally developed at Dartmouth College and stands for Beginner's All-Purpose Symbolic Instruction Code. It's ease of execution by the use of English-like commands make it the most popular micro-language available today. Your SANYO computer, with its advanced SANYO-BASIC abilities will stand as a ready friend to do your every software task.

Setup

Let's get started using the language and getting familiar with it. We must first "boot-up" our system. Follow these procedures:

1. Place your Work Disk into your SANYO machine after powering-up your system (A drive for dual drive systems).

2. The system will self-load the CP/M Operating System and respond with the CP/M prompt message:

   A>

3. Type the letters **BASIC** and follow this with striking the RETURN (CR) key. You will hear the system access the disk.

4. You will then see the SANYO-BASIC message appear and the SANYO-BASIC Prompt character:

   >

5. We are now ready to begin working with SANYO-BASIC!

6. We are now in the SANYO-BASIC Operating System. CP/M based calls to disk will not work.

Display

Note the flashing line on the screen. That's called a CURSOR. It can be in any of several hundred locations on the screen. Actually the screen is a matrix of character positions that is 80 wide by 25 deep. TheCURSOR'scurrentpositionis where any entrybyyouon the keyboard will show itself.

ENTER	DEPRESS	RESULT
PRINT 25 + 75	RETURN	100

We are executing a statement in what the language interprets as the "DIRECT EXECUTION MODE". The computer knows you are through with a command when you depress the RETURN (CR) key. It then "looks" at what you have told it to do and will attempt to do so

The keyboard is our way of communicating to the computer what we want it to do.

The computer will process our instruction after we depress RETURN (CR).

The computer will interpret text between the prompt character > and the Carriage Return (CR) depressment as the instruction we want it to do.This is called the DIRECT EXECUTION MODE.

We can give the computer a sequence of things to do by placing colons between the instruction. An example:

```
> PRINT 3+3: PRINT 45: PRINT 75+10      (CR)
```

(CR) means press RETURN Key

We can ask the computer to PRINT something that has characters and letters .....for example the words "SANYO 4000 Series".

```
> PRINT"SANYO 4000 Series"
```

Here anything between the " " (quote marks) will be PRINTed.
Note that we cannot print the quote marks as the computer will not know where to end!

In each case our SANYO computer has performed the small task we have asked it to do and has returned back to us a prompt ">" character, telling us it is ready for its next task. In virtually every case this prompt character signifies a READY status. LOOK FOR IT!

Syntax

We have been giving the computer a PRINT statement to process. The SANYO-BASIC list of known instructions is quite long and is the subject of Chapter 9. The point is; there is a limit to what SANYO-BASIC knows as commands. If you give the computer a command it is not familiar with, or if you give it a command in the wrong format you will see a message to you stating that you have made a "SYNTAX ERROR". The computer will ignore the command and go back to the prompt mode awaiting the next instruction. PLEASE REVIEW CHAPTER 9 prior to writing any large programs!

Listings of Commands

At this point we have been giving the computer only one line worth of instruction. The structure of SANYO-BASIC is such that we can give the machine many, many lines of commands.....as long as we label each line with a number so that the computer will know which one to process first. To do this we can place a number in front of each command. Type in the following:

```
> 10 PRINT"HELLO"      (CR)

> 20 PRINT"MY FRIENDLY"      (CR)

> 30 PRINT"COMPUTER"      (CR)

>
```

Now after the prompt character type the word LIST followed by a (CR) The computer responds with a LISTing of what you have entered.

```
> LIST
```

These LISTings of a program are important for diagnostic purposes or for awareness of structure. There are several modes of LIST and these follow in the table below;

LIST xx	LISTs a single line number xx.
LIST - xx	LISTs from the start to line number xx.
LIST xx -	LISTs from line xx to the program's end.
LIST xx - yy	LISTs from line xx to line yy.

In each of the above the pressing of the BREAK key will discontinue the LISTing of the program. The numbers xx and yy used as variables in the above table can range from 1 to approximately 64000.

The pressing of the Space Bar will temporarily hold the listing in a pause mode until the Space Bar (or any other key) is depressed again.

Running a Program

Once a program resides in memory in a numerical order one may RUN or actually execute it. The computer will start at the least ordered line in the program (indicated by the line number) and will execute each line of instruction exactly as it was written. SANYO-BASIC is an interpretive language. This means each instruction is broken apart into sub-sets and executed during each RUN phase.

Let's RUN the program just entered and LISTed in the previous section. To do so first LIST the program to make sure it is still there.

Now let's tell it to actually RUN the command sequence. To do this simply type in the word RUN after the prompt character and depress the (CR) key.

Your output on the screen should look something like:

```
HELLO
MY FRIENDLY
COMPUTER

>
```

You have just RUN your first program!

The RUN command has the following alternate mode:

RUN xx RUN the program starting from statement # xx

This is useful for the execution of parts of the program.

Clearing a Program

You may wish to clear away your program to make room for another. To do so simply type in the NEW command after the cursor. This will clear all program statement numbers, clear all variables to zero and allow for a new program entry sequence. Do not confuse this with the CLEAR command described in Chapter 9.

Ending a Program

The last statement of a program can optionally be an END statement. This will close all files and permit the program to return to the SANYO-BASIC prompt mode. For more details see Chapter 9.

Variables

Variables are names used to represent program values used in the execution of a particular program. Variables are labeled. An example of a variable is the label VAR . You may specify a value for the variable:

```
VAR = 3.1234
VAR = 4
VAR = 32 + 16 + 7 + 45.6
```

Or, you may let the variable assume different values based on other variables within the body of the program:

```
VAR = DATA1 + 3
VAR = 3.1456 + BAL + OLD
VAR = VAL1 + 1
```

Before any variable is assigned it has a value of 0.

Variable names can be of a length suitable for your description, however, only the first two letters of the variable are recognized by SANYO-BASIC. This means the following variables are considered different:

```
BALANCE         BREAK        BUCKS
```

And these are considered as the same:

```
BALANCE         BALL         BACKTAX
```

A variable cannot be one of the special RESERVED WORDS which are indicated in the reference section. If they are a miss-interpretation may occur. A variable may also not contain a reserved word within in. BLIST, for example, is illegal.

The first character of a variable must always be an alpha character.

Variables may be either numeric values or strings. Strings are written with a special symbol $ after them to indicate they are alpha-numeric values. An example of a string variable is;

```
B$ = "THIS IS A STRING EXAMPLE"
```

Note that string declarations are always in quotes.

Numeric variables may declare INTEGER, SINGLE or DOUBLE PRECISION values. These are shown in the table that follows:

```
INTEGER VALUE           % terminator    example: DRAT%

SINGLE PRECISION        ! terminator    example: HO!

DOUBLE PRECISION        # terminator    example: V#
```

If one omits the type declaration single precision is assumed.

Arrays

Many times it is necessary to use many variables with subscripts. One can think of them as elements of an array. An array is a grouped set of variables that are first DIMENSIONed using a DIMENSION statement. Array variables have the same number of subscripts as there are dimensions in the array. For example a one-dimensional array has varaiable subsripts indicated by

XONE(x) Where x is the subsript. Example. XONE(23)

A two dimensional array has two subscripts:

XTWO(x,y) Where x and y are subsripts. Example XTWO(10,20)

and so on to other dimensions.

Notice that subscripts must always be integers. A DIMENSION statement as described in Chapter 9 sets the limits for the array and defines the working allocation for it.

Transfer of a Program Execution

As a program is RUN the execution of it may be transferred to a different spot by use of the GOTO and GOSUB commands.

GOTO

This command is followed by a line number and will transfer the execution of the program to the new line specified. The line specified must be in the program body or an ERROR will occur. An example follows:

```
10      PRINT"THE VALUE IS:"

20      PRINT A

30      A=A+1

40      GOTO 10
```

This program will start printing the value of A from 0 on up continuously until the BREAK key is depressed. It is a good example of an ENDLESS program. Avoid such "loops" as they must be "broken" by human action.

The GOTO command may be nested within a conditional command. This means a decision can be made as whether or not to process the GOTO :

```
IF XYZ THEN GOTO 100
```

This command will transfer control to 100 only if the tested condition is true.

GOSUB

A very powerful transfer can be effected using the GOSUB command. This command stands for GO TO SUBROUTINE. A Subroutine is a piece of program that can be "called" from your main program. It may perform a special thing as defined by the program body and may be "called" from several places in the program. Let's examine the flow of such a "call":

```
10      ...
15      ...
20      ...
25      GOSUB 300     Here the execution jumps to #300
30      ...           Return here
40      ...

                                  300     ...
                                  310     ...
                                  320     ...
                                  330     RETURN
```

Note the use of the required RETURN statement of get us back to command line 30.

A GOSUB call is powerful as it may be executed from any part of the program. In fact, subroutines may be "nested", meaning they may themselves call other subroutines. Many lines of program can be saved by using subroutines to handle repetitive tasks.

Summary

This chapter served as a brief introduction to the powerful SANYO-BASIC program set that is the readily available system language for your SANYO. Chapter 8 and Chapter 9 will define in more detail the command structure and general execution parameters. Use the syntax carefully.

CHAPTER 8 - MATH AND LOGICAL RELATIONSHIPS

Introduction

Mathematical functions are constantly used in a computer. It is the intent of this chapter to present an overview of SANYO computers' ability to handle, store and process numbers and functions.

Mathematical Operators

You can process numbers in mathematical functions by knowing how, symbolically, various operators are used in the computer. The following table will help.

OPERATION PERFORMED	SYMBOL	EXAMPLE
ADDITION	+	A+B
SUBTRACTION	-	B-A
DIVISION	/	B/A
MULTIPLICATION	*	A*B
POWER EXPONENTATION	^	A^2
NEGATION	-	-A
INTEGER DIVISION	\	A\B

These operators establish the format for math functions.

Use of Parentheses in Expressions

To be certain of a correct function process place quantized portions of your mathematical expression or function in parentheses. There is a sequence and priority of process which will cause expressions to be evaluated differently. Make sure your expression is processed correctly by parentheses format.

EXAMPLE: (23 * 76) raised to the 14th power would be written as;
(23*76)^14

EXAMPLE: 100 subtracted from 500 and the answer multiplied by the 5th power of 16 would be written as;
(500-100)*(16^5)

When you use parentheses notation BALANCE YOUR PARENTHESES! There should be as many "("s as there are ")"s in a notation. A system error will result if this rule is not followed.

Priority of Sequence in Operator Processing

The various operators are processed in a set sequence of priority. This means one has to be very aware of the sequence unless one uses the parentheses notations as described above. The following is the sequence of operations for functions:

1. Calculations in Parentheses evaluated first.

2. Power (Exponent) Calculations.

3. Negation of a number. (This is not subtraction)

4. Multiplication and Division

5. Integer Division

6. Addition and Subtraction

Operational priority is performed left to right in an expression if it has a series of operators present. As an example consider the expression:

4*(6+4)*2/2+1-3

The answer will be 38. Start from the left and observe priority.

Relational Operators

We may wish at times to compare two values in an expression or elsewhere in our program. To do so we have the ability to use RELATIONAL OPERATORS. These symbols equate two values or expressions. The following are SANYO relational operators:

RELATIONAL OPERATOR	SYMBOL	EXAMPLE
EQUALITY	=	X=Y
INEQUALITY	<> or ><	X<>Y
LESS THAN	<	X<B
GREATER THAN	>	B>X
LESS THAN or EQUAL TO	<= or =<	A<=B
GREATER THAN or EQUAL TO	>= or =>	B>=C

These operators compare the two quantities and the result is either true (indicated by a "1") or false (indicated by a "0"). The operators can nest inside other commands for comparisons;

Example: IF(SIN(X)>0) GOTO 1000

IF(DATA<100.45) THEN DATA = 1.2345

Please use extreme caution when using the EQUALITY operator. Equality in a computer is an extremely accurate thing. One may be off by a factor of one millionth and the equality will not be there.

Please note that when relational operators are combined with arithmetic expressions in a comparison the arithmetic is always done first under the priority rules mentioned earlier.

Logical Operators

Logical operators are implemented in the structure of every computer. Many times it is desirable to handle numbers in a binary format. SANYO has provided a way to do this with LOGICAL OPERATORS.

Logical operators work with binary numbers. Conversion routines exist in the SANYO to efficently handle binaries. The reference section provides a guide for conversion of decimal to binary numbers.

Logical operators work by converting their operands to sixteen bit, signed, two's complement integers in the range -32678 to +32767. An error will result if your operand is greater than these limits. Let's look at some examples:

10 = 1010 (actually stored as 0000000000001010) = A

31 = 11111 (actually stored as 0000000000011111) = B

The AND operator states the result is true only if both operands are "1"s.

A AND B = 01010 = 10

The OR operator states the result is true only if one OR the other is a "1".

A OR B = 11111 = 31

The XOR operator states the result is true if one or the other is a "1" but not both.

A XOR B = 10101 = 21

The NOT operator inverts every bit.

NOT A = 0101 = 5

The EQV operator states the result is true if both are the same state, "1"s or "0"s.

A EQV B = 01010 = 10

The IMP operator states the result is true by implied result. The first operator being true implies the second.

A IMP B = 1111111111111111 = -1
B IMP A = 1111111111101010 = -22

As with arithmetic operators there is a priority sequence. For logical operators this is the sequence.....

1. NOT
2. AND
3. OR
4. XOR
5. IMP
6. EQV

Negative numbers are the two's complement of the binary value with the most significant bit set. The count is "backwards" as in standard signed integer form.....

```
EXAMPLE:    -1    =    1111111111111111
            -2    =    1111111111111110
            -7    =    1111111111111001
            -30   =    1111111111100010
```

NOTE: The two's complement of any number is the bit complement plus 1.

One may use logical operators to test bytes for a given bit pattern. The AND operator, for example, can be used as a "MASK" to select a bit or series of bits in a particular byte. The OR operator can be used to "MERGE" two bytes together. The XOR operator can be used to compare two bytes....and so on.

Operators Onto Strings

Strings can be looked at and compared with relational operators. For example, if we wished to compare A$ to B$ we may use:

```
A$ > B$          A$ <> B$
A$ < B$          A$ <= B$
A$ = B$          A$ >= B$
```

Strings may also be combined using the + operator only. This process is called CONCATENATION.

EXAMPLE:

```
A$ = "TOM AND I "
B$ = "WENT HOME AFTER WORK"

A$ + B$ = "TOM AND I WENT HOME AFTER WORK"
```

Accuracy in Expressions

The SANYO computer series will evaluate expressions in either single precision or double precision modes. Even so you should be aware that the final result is only as accurate as arithmetic routines can make it. For EXAMPLE:

```
2  is not equal to 1.999999999999999
     and never will be.

Some functions give you a result that is
that close.

     BE CAREFUL!
```

CHAPTER 9 - SANYO-BASIC

Introduction

SANYO-BASIC is a language processing program to meet the most extensive implementations of the BASIC interpreter language. Developed for SANYO computers, it provides a wide range of applications in the fields of office, home and technological computations. A great number of instruction words, various math functions and an extensive program editor are available in SANYO-BASIC.

This section describes the operations of instruction words, and program editing of SANYO-BASIC.

No abbreviations are allowed in SANYO-BASIC (BASIC).

Symbols for Syntax Definition

The following is a guide for formats used in this chapter:

1. Items enclosed in square brackets ([ ]) may be omitted. However when omitted operations may be different, therefore a clear understanding of each instruction is necessary.

2. Items in braces ({ }) can be either omitted or repeated at any time.

3. Items in angle brackets (< >) cannot be omitted.

4. When items are separated by a vertical bar (|), a single item must be chosen. When both of the optional items are underscored, additional choices must be made.

5. All characters and symbols appearing in general statements and commands have particular meanings. They must be written as they are:

<	Less than sign
>	Greater than sign
=	Assignment symbol or equal sign
?	Question mark
@	At-sign
[	Left square bracket
]	Right square brcket
/	Back slash or interger division symbol
^	Exponentiation symbol
_	Underscore
{	Left brace
}	Right brace
X	Tabulation set sign
~	Wavy sign or "TO" sign

The commands and statements of the programs are described with upper-

case letters, lower-case letters, and symbols arranged with proper syntax. The upper- and lower-case letters are equally handled and interpreted to describe the commands and statements; however, the lower-case letters do not usually match program listings as SANYO-BASIC is commonly written in upper-case.

Operation Modes of SANYO-BASIC

INITIALIZATION

The following line must be typed while in the Operating System Mode (such as CP/M) to initialize the SANYO-BASIC interpreter:

A > SANYO-BASIC [<"file name">] [/F:<: the number of data files to be used>] [/M:<maximum address.] <CR>

When the line has been typed, an interpreter is loaded from the system disk into the memory and the control enters the BASIC Mode.

In detail:

< "File name" >

Specifies the name of a disk file in which a program is stored. When specified, programs to be executed after the initialization of the BASIC interpreter are automatically loaded and executed after the initialization. If the file name is omitted, the control enters the BASIC Mode but waits for keyboard entry while displaying a Prompt. For program files including file names, see the File I/O section of this manual.

< The number of data files to be used >

Defines the maximum number of data files which a program with the specified file name can open at a time. When omitted, the default value "3" is assumed. In SANYO-BASIC a maximum of 15 data files may be opened at a time. (The buffer area of a disk file is reserved when the number of data files is specified.)

< Maximum address >

Sets the end of memory where a SANYO-BASIC program is loaded. It is necessary to specify <maximum address> when the user wishes to define the size of memory used in SANYO-BASIC for assembly language subroutines. When <maximum address> is omitted, the system at the time of initialization defines the maximum address and displays the number of bytes in user's area. The number of bytes will not be displayed if a file name has been specified to be run.

```
A> BASIC  <CR>
A> BASIC  "PROGRAM-1"  <CR>
A> BASIC  "PRGM-1"/F:5 <CR>
```

PROMPT AND CURSOR

In SANYO-BASIC mode a "wait" state is entered after the initialization of the SANYO-BASIC interpreter, requiring the key entry of Sanyo-BASIC instructions or other directive commands. In the SANYO-BASIC mode, a prompt character (>) is displayed.

```
Example:    Ready
            >          Cursor (blinking)
```

When the Prompt is displayed, a SANYO-BASIC instruction or program can be keyed in.

DIRECT MODE

When a Prompt is displayed and a program statement is keyed in without a line number in front of it but with a carriage return <CR> at the end, the statement will be implemented and executed immediately. This is called the Direct Mode. The operation's result of the keyed-in statement implemented in the Direct Mode is stored in memory, but the instruction itself is deleted. An execution of multiple statements in the Direct Mode is also available by placing colons (:) between statement.

```
>A = 5 * 5: PRINT A <CR>
16    (this is the answer)


>PRINT "SANYO MBC-SERIES." <CR>
SANYO MBC-SERIES
```

PROGRAM EDIT MODE

The Program Edit Mode is used to key in, edit, and store into memory a SANYO-BASIC source program; the method of editing is described below. (This section describes a simple outline, and for the details of program editing see the "PROGRAM EDITING" section of this manual.)

Line Editing

When a line number and then a statement are keyed in, they are loaded into memory as a unit of source program. The line number may be created automatically by use of the AUTO command. The renumbering operation to modify line numbers of source programs is performed by the RENUM command. The string of "line number statements" is keyed in repeatedly so that the source programs are stored in sequence in memory for later use by entering the RUN command. Please see command EDIT.

RUN MODE

The source programs, created in the Program Edit Mode and stored in memory, are executed in sequence by the line numbers entered. The RUN command followed by <CR> does this. The execution can be halted by pressing a BREAK key or when the STOP statement is executed in the program. It can be restarted by use of the CONT statement. If during the Halt state modifications are made in a source program or an error occurs during Direct execution, the program can not be continued. The Run Mode returns to the SANYO-BASIC Mode when the END statement is performed during the execution of the program or when the source program has run out of executable statements. However, when the SYSTEM command is entered and executed in the program the control returns to the Operating System Mode.

RETURN TO OPERATING SYSTEM MODE

The control returns from the SANYO-BASIC Mode to the Operating System Mode when the SYSTEM command appears in the program during program execution or when the SYSTEM command is directly performed in the SANYO-BASIC Mode. Whenever a CTL key and a RESET key are pressed simultaneously, the control returns to the Operating System Mode if a system disk is installed.

The changeover from the Operating System Mode to the SANYO-BASIC requires the BASIC initialization procedure described earlier in this section. (When a SANYO-BASIC initialization procedure is performed again, the parameters may include the number of data files to be used. The maximum address, which as been specified at the time of initialization, can again be modified.)

Line and Statement Use in a Program

CHARACTER AND SYMBOLS USED IN PROGRAMS

The following characters and symbols are used in SANYO-BASIC:

26 upper-case letters	ABCD. . .
26 lower-case letters	abcd. . .
10 numerics	0123456789
32 symbols	

!	(Exclamation point)	;	(Semicolon)
#	(Number sign)	$	(Dollar sign)
%	(Percent)	&	(Ampersand)
(	(Left parenthesis)	*	(Asterisk)
)	(Right parenthesis)	+	(Plus sign)
-	(Minus sign)	,	(Comma)
:	(Colon)	<	(Less than)
?	(Question mark)	>	(Greater than)
@	(At sign)	[	(Left bracket)
_	(Underscore)	]	(Right bracket)
{	(Left brace)	"	(Double quotation mark)
}	(Right brace)	'	(Single quotation mark)
.	(Period/decimal point)	^	(Up arrow/exponentiation)
\	(Backslasher/integer)	/	(Division symbol)
=	(Equal sign/assignment)		(Space)

Lowercase letters are converted to upper-case letters for storage in memory. Othercharactersare available in a programmable format. If the lower-case lettersare regarded as strings (literal data),they are stored asthey are. Usable characters correspond to the internal codes of the SANYO MBC-4000 series and are displayed on the screen.However, the amount of usable characters that will be accepted by a printer is less.For printing out programs we suggestdetermining the range of printable characters by the printer andusing only those in the program.

A character is written into the program by pressing the corresponding key on the keyboard. Special characters are created only by the multiple use of keys. In the SANYO-BASIC programs the special characters in Columns 0 and 1 of the character codes table referenced at the back of this manual cannot be keyed in through the keyboard. They are written in the programs only by the use of CHR$ functions.

```
10 PRINT CHR$(07)
   (Rings Bell)
20 PRINT CHR$(89)
   (Prints Y)
```

LINES AND LINE NUMBERS

A line is the minimum unit of a program consisting of a line number, a group of statements (descriptions of instructions), and a <CR> (to terminate a line). The general format of a line is as follows:

<line number> <statement> {:<statement>} <CR>

```
>10 A = 3:B = 100:C = A + B:PRINT C <CR>
>RUN
 103
```

* A line number is a five-digit positive integer in the range from 0 to 65535. (Two bytes are allocated for a line number in memory.) The leading zeros are ignored.

No spaces can be included in the numeric parameter to show a line number.

```
>00100  A = 3   (Same as 100, leading zeros
                 ignored)
>1 1 0  A$ = "THREE"  (Erroneous spaces)
>120 PRINT A, A$
>130 END
>RUN
 Syntax error in 1
```

* In SANYO-BASIC the carriage return and line feed are keyed in by use of the keyboard's <CR> key, and the line feed by the <LF> key. Both functions are visible on the display screen. The <CR> key is used to terminate a logical line, and the <LF> key, on the other hand, is used to continue typing a logical line on the next physical line. The keying-in of data during an operation is terminated by use of the <CR> key or the <ENT> key.

A line number indicates the index for program operation control and is also used for line search in program editing.

<Statement> is an instruction described within the SANYO-BASIC

syntax rules and is also the minimum unit or element to compose a program.

Colon (:) is a delimiter placed between one statement and another in composing several statements into a single program line. A program line composed of a single statement is called "Simple Statement", and one composed of a several statements, which are delimited with colons, is called a "Multiple Statement".

<CR> indicates the end of a logical line and is entered by use of the <CR> key.

The length of a line cannot exceed 255 characters including the line number,

<LF>, and spaces. With a <LF> the lines on the screen or in the printer are scrolled, but every physical line begins with <line number> and ends with <CR>.

KEYWORDS AND SPACES

The commands, statements, and function names are the **key words** in SANYO-BASIC and do not include spaces between letters. In addition, the keywords, variables, constants, and logical operators must be separated by spaces, enclosures, numeric operators, or other delimiters syntactically recognized. The use of spaces in other cases is at the programmer's option and is ignored by the system.

```
>10 P R I N T     (Erroneous)
>20 A = B + C     (Valid)
>30 A = B AND C  (A is assigned as a logical AND
                   operation of A and B
>40 A = BANDC
>RUN
 Syntax error in 10
```

DATA

Data Types

Data is written with numbers or characters for processing in the program. Data with numbers is regarded as "numeric data" and the data with characters as "string data". Either a constant or variable can be assigned as a data item. The variables are of two types: i.e., simple variables and subscripted variables (an array). The numeric data is divided into three categories; namely integers, single-precision real numbers, and double-precision real numbers.

Examples of data are shown in the following table:

	String	Integer	Numeric
CONSTANTS	"VALUES" "Your name?"	245 32	3.678 1234.567 -3.45E35 .345!
SIMPLE VARIABLES	A$ SUM$	ABC% K%	DAT H!
ARRAY VARIABLES	B$(10,4,8)	X%(N)	Z!(I,J)
MEMORY USAGE	1 byte/char.	2 bytes	4 bytes or 8 bytes
OCTAL NUMBERS	See Technical Reference Section		&7632

HEXADECIMAL NUMBERS	See Technical Reference Section &H3AF4

Constants

1. Integer constants

Nunmbers between -32768 and +32767 with no decimal point or fraction.

```
>10 A% = 365
>20 B% = -12345
>30 PRINT A% + B%
>40 END
>RUN
 -11980
```

2. Real numbers (Single-precision) 6-digit significant numbers with an exponent between -64 and +63, which are expressed in the following three ways:

(i) Number of up to 6 digits expressed as + XXXX.XX

Example: 1234.56

(ii) Number with an exponent (E) expressed as
+ .XXXXXXE+XX

Example: 123456E-05

(iii) Number with a suffix "!" written as + XXX.XXX!

Example: 123!

3. Double-precision constants (Double-precision real numbers) 14-digit significant numbers with an exponent between -64 and +63, which are expressed in the following three ways:

(i) Number of up to 14 digits expressed as
+ XXXXXXXXXX.XXXX

(ii) Number with an exponent expressed as
+ .XXXXXXXXXXXXXXD + XX

(iii) Number with a suffix "#" written as + XX.XXXX#

4. Octal number

Octal numbers in the range between 0 and 177777 with a prefix "&" or "&O"

Example: &175432
&O324

5. Hexadecimal number

Hexadecimal numbers in the range between 0 and FFFF with a prefix "&H"

Example: &H256
&HA4F5

6. Character constant

Characters enclosed in quotation marks (" ") within 255 characters.

```
10 A$ = "SANYO "
20 B$ = "MBC-SERIES"
30 PRINT A$;B$
40 END
```

The double quotation marks (" ") cannot be used within a string constant.

```
10 C$ = ""SANYO-MBC SERIES""
20 PRINT C$
>RUN
Syntax error in 10
```

Quotation marks may be printed by using the CHR$(34) symbol.

```
10 C$ = "SANYO-MBC SERIES"
20 PRINT CHR$(34); C$; CHR$(34)
>RUN
 "SANYO-MBC SERIES"
```

Variables

1. Variable name

Variables are the changeable values in a program and are referred to by specifying their names. The variable names are written with an alphanumeric string of optional length beginning with an alphabetical character. The first 16 characters indentify the variable.

```
10 TR = 10
20 TRE = 20
30 TRANS = 30
40 PRINT TR, TRE, TRANS
>RUN
 10            20            30
```

The variable names must not be spelled the same as keywords, but may include their spelling. Any variable name beginning with FN or USR is erroneous.

```
10 AND = 100   (Erroneous, AND is an operator)
20 BAND = 0        (Acceptable)
30 AFN10 = A + B   (Acceptable)
40 FNABC = 1       (Erroneous, FN is an operator)
50 BAFN = 1        (Acceptable)
```

2. Types of variables

The variables are classified into strings or numerics (integers, simple-precision real numbers, and double-precision real numbers).

The type of data is declared by the DEF statement or by attaching a suffix at the end of the variable as a type specification symbol.

Type of variable		Symbol	Example
Numerics	Integer	%	A%, INTEGER%
	Single-precision real number	!	RO! REAL!
	Double-precision real number	#	B5#, DBL#
Characters	(String)	$	LITERAL$

When a variable has no type specification symbol or is not declared by the DEF statement, a single-precision real number is assumed. (The default type for a numeric variable name is single-precision.)

3. Arrays (Subscripted variables)

Arrays are declared by the DIM statement before they are used. There is no limitation on an array's dimension or number of elements. However, the whole length should fit in the total storage capacity.

An array which consists of 10 or fewer members may be used even if not declared by the DIM statement. (The declaration is assumed to have been done when the array is used for the first time.)

Expressions and Operators

An operator is a symbol to show the operational method (i.e. adding, subtracting, or comparing the date). A combination in which some number of data items are related by operators according to syntactic rules is called an expression. One expression produces one value as a result of the data opera-

tion performed, in the order that the operators specify and the priority of the operator. We call that value, in this manual, an evaluation result which naturally shall be either numeric or string data.

ARITHMETIC OPERATORS

Arithmetic operators are divided into the following eight types:

	OPERATOR		EXAMPLES	
1	^	Raising to a power (exponentiation)	A ^ B	Raise A to the Bth power
2	-	Minus sign	-A	Minus A
3	*	Multiplication	A * B	Multiply A by B
4	/	Division	A / B	Divide A by B
5	\	Integer division	A \ B	Divide A by B, and discard the decimal part.
6	MOD	Remainder	A MOD B	Remainder of A divided by B **(Both A and B are rounded off to integers before being divided.)**
7	+	Addition	A + B	Add A and B
8	-	Subtraction	A - B	Subtract B from A

ARITHMETIC EXPRESSIONS

An arithmetic expression is a combination of numeric data items and arithmetic operators, generally written as follows:

<data item> {<arithmetic operator> <data item>}

The data items to be written are classified below:

1. Numeric constant
2. Numeric variable
3. Numeric function
4. (Arithmetic expression)

The operation of an arithmetic expression is performed in the following order according to its priority as shown below. Sequence is important.

Order of Precedence:

1. Calculations enclosed in parenthesis are always evaluated first and have the highest precedence.
2. Power calculation or exponentiation.
3. Minus sign or negation.
4. Multiplication and division.
5. Integer division.
6. Remainder.
7. Addition and subtraction.

The calculations which have the same operational priority are performed from left to right.

```
 10 A = 2*(2+1)*2/3+1-2
 20 PRINT A
>RUN
 3
```

RELATIONAL OPERATORS

Relational operators are used to compare one data item to another. They are divided into the following six types:

1) = Equal?	A=B	Is A equal to B?
2) <> Unequal?	A<>B	Is A unequal to B?
3) < Less than ?	A<B	Is A less than B?
4) > Greater than?	A>B	Is A greater than B?
5) <= Less or equal?	A<=B	Is A less or equal B?
6) >= Greater or equal?	A>=B	Is A greater or equal B?

RELATIONAL EXPRESSION

A relational expression is a combination of data items and relational operators and is generally written as follows:

<data item> {<relational operator> <data item>}

The data item is an arithmetic expression or character string. Data items written in relational expressions must be the same type; i.e., arithmetic expressions or character strings. (Arithmetic expressions cannot be compared with string data or vice versa.)

When the data items are related to each other with relational operators, the evaluation result will become -1 if the expression is affirmed (true) and if not affirmed (false).

Comparison of the numeric values is performed in two ways-- whether or not one value is smaller/greater than the other.

The size factor determines the comparison of two numeric values, that is, the comparison of numeric values is performed by determining whether or not one value is smaller/greater than the other.

The comparison of character strings is performed by the following procedure:

1. The value of one character is determined in accordance with the table of ASCII character codes.

2. The value of strings having more than two characters are compared as follows:

The value of comparison based on Rule 1 above should be done on the first pair of characters, then on the second, and so forth.

The string in which a character with a smaller code value first appears is determined as smaller.

If either string ends but determination is not made yet, the shorter one is regarded smaller.

Example:

COMPARISON	RESULT	
"A" > "B"	-1 TRUE	(ASCII value 65 is less than 66)
"?" < "="	0 FALSE	
"ABC" < "ABCA"	-1 TRUE	
"ABA" > "AA"	-1 TRUE	
"XYZ" < "ABC-1"	0 FALSE	

If more than two relational operators are included in an expression, the operation is performed from left to right under priority rules.

LOGICAL OPERATORS

Six logical operators are described below in order of operational priority. (See the technical reference section for further explanation.)

Order of Precedence:

1.	NOT	Negation
2.	AND	Logical product
3.	OR	Logical sum
4.	XOR	Exclusive logical sum
5.	IMP	Implication
6.	EQV	Equivalent

LOGICAL EXPRESSION

A logical expression generally has the following forms:

[NOT] <data item> {<logical operator> [NOT] <data item>}

Either an arithmetic expression or a relational expression can be written as the data item. In case of an arithmetic expression, its evaluation result should be an integer between -32768 and +32767. If the evaluation result were a real number, it would be rounded off. If it falls out of the range an error occurs.

A logical expression evaluates the result with the following procedure:

1. When more than two logical operators are included:

   a. Execute the processing in accordance with the priorities.
   b. If the two different logical operators have the same priority, execute from left to right.

2. When the value of <data> is assigned with either -1 (true) or 0 (false), the evaluation of the logical expression will become -1 or 0.

3. When an integer is included in the evaluation result of <data item>, the logical operation is performed with binary notation (binary integers are allocated for each bit) regarding the result as a binary number. The evaluation result will be converted into the integer.

Example:

1. Definition of logical operators

NOT	X	NOT X
	1	0
	0	1

AND	X	Y	X AND Y
	1	1	1
	1	0	0
	0	1	0
	0	0	0

OR	X	Y	X OR Y
	1	1	1
	1	0	1
	0	1	1
	0	0	0

XOR	X	Y	X XOR Y
	1	1	0
	1	0	1
	0	1	1
	0	0	0

IMP	X	Y	X IMP Y
	1	1	1
	1	0	0
	0	1	1
	0	0	1

EQV	X	Y	X EQV Y
	1	1	1
	1	0	0
	0	1	0
	0	0	1

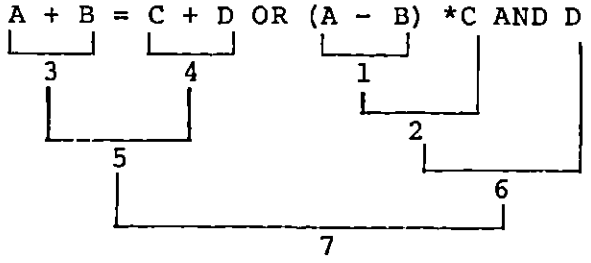
2. Logical operation including integer(s)

```
          15 AND 16
                 15      ->        01111
                 16      ->        10000

          15 AND 16      ->        00000    ->      0

          15 AND 10
                 15      ->        1111
                 10      ->        1010

          15 AND 10      ->        1010     ->     10

          -1 AND 16

-1 ->    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
16 ->    0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

-1 AND 16 ->0 0 0 0 0 0 0 0 0 1 0 0 0 0 -> 16

           8 OR 16
                 8       ->        01000
                16       ->        10000

           8 OR 16       ->        11000    ->      24

          15 OR 16
                 15      ->      1 1 1 1
                 16      ->    1 0 0 0 0

          15 OR 16       ->    1 1 1 1 1    ->      31
```

EVALUATION OF COMBINED EXPRESSIONS

A relational expression and an arithmetic expression can be written as a data item in a logical expression, and an arithmetic expression can be written as a data item in a relational expression. So, a combination of different expression is performed in the order of 1) arithmetic expression, 2) relational expression, and 3) logical expression.

When A=1, B=2, C=2 and D=1, the evaluation result of the following expression will become -1. With the evaluation following these steps:



It is noted that "=" used in the LET statement and "=" as a relational operator are different in meaning.

```
10 LET A = 2 = 3    (0 is assigned to A)
20 LET C = B = 1    (-1 is assigned to C when B=1
                      0 is assigned to C when B=0)
```

STRING EXPRESSION

A STRING EXPRESSION combines one character string with another and is written in the following format:

<data item> {+ <data item>}

One of the following three can be written as a data item:

1. String constant
2. String variable
3. Function which results in a string (Character functions)

The evaluation result of a string expression is also a character string with less than 255 digits. When the string length exceeds the limit an error occurs.

```
10 A$ = "SANYO "
20 B$ = "M"
30 C$ = "B"
40 D$ = "C"
50 E$ = "-SERIES"
60 PRINT A$;B$;C$;D$;E$
>RUN
SANYO MBC-SERIES
```

INSTRUCTION WORDS

The SANYO-BASIC instruction words perform direct executions and are written in the user programs. Generally speaking, in SANYO-BASIC there is no descrimination between the commands and the statements, however, with some instruction words the control does not proceed to the next step, but returns to the BASIC mode even if an execution is completed.

The instruction words are explained in an alphabetical order but some interrelated instruction words intentionally disregard this order.

The following instruction words return control to the SANYO-BASIC mode.

AUTO
DELETE
END
LIST/LLIST
LOAD (without R option)
MERGE
NEW
RENUM
STOP (The program can resume by the CONT.)

The control returns to the Operating System Mode after this instruction word is executed. SYSTEM

AUTO

Purpose

The AUTO is used to create line numbers automatically in the line edit mode at the time of program writing.

Form

AUTO [<line number> [,[<increment>]]]

Description

(1) When <line number> specification is made, the system displays the specified line number and waits for the key-in of a statement. When the statement is keyed in and <CR> is pressed, the system displays a new line number, which is the current line number plus the increment and waits for another statement to be entered. The same process is repeated.

(2) When the line number specification is omitted, the creation of line numbers starts from 10. When the specification of <increment> is omitted, 10 is assumed. (The line numbers go 10, 20, 30 and so on.)

(3(When the line number is specified as <line number>, only the previous increment (if one has been used) is designated.

(4) The AUTO mode is stopped by pressing the CTRL + C and the control returns to the SANYO-BASIC Mode. When the line number exceeds 65535 or when a line edit error is found in the keyed-in statement, the system displays an error message and returns the control to the SANYO-BASIC Mode (See appendix B).

(5) When 0 is assigned to <increment>, an error message is displayed.

Example

```
> AUTO    100,   20
> 100
> 120
> 140
> 160

> AUTO.      (The final line number of a program
              which is stored in memory will be
              displayed.)
```

BEEP

Purpose

The BEEP sounds a buzzer.

Description

(1) The buzzer is generated for 0.4 seconds.

Form

```
BEEP
```

Example

```
>10 REM TEST FOR BEEP
>20 PRINT "SANYO!"
>30 BEEP
>40 END
>RUN
```

CALL

Purpose

The CALL statement calls the subroutines written in Assembly language from the SANYO-BASIC program. To call the subroutines written with SANYO-BASIC, the GOSUB statement is used.

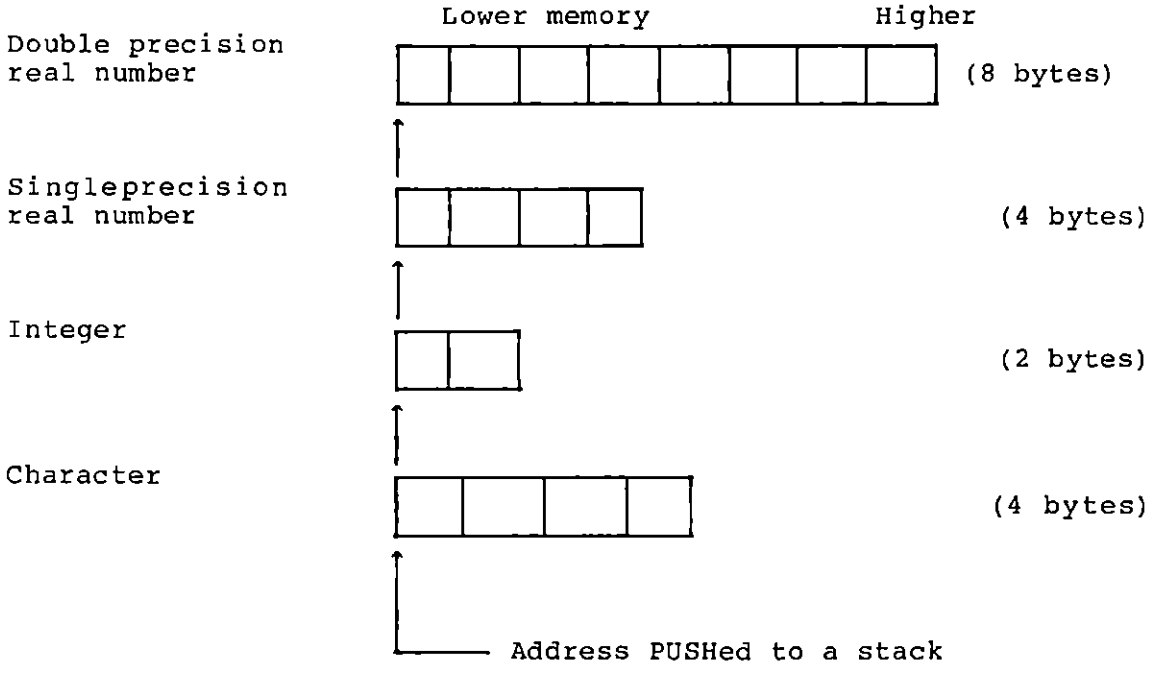
Form

```
CALL  <start address>, {<argument>}
```

Description

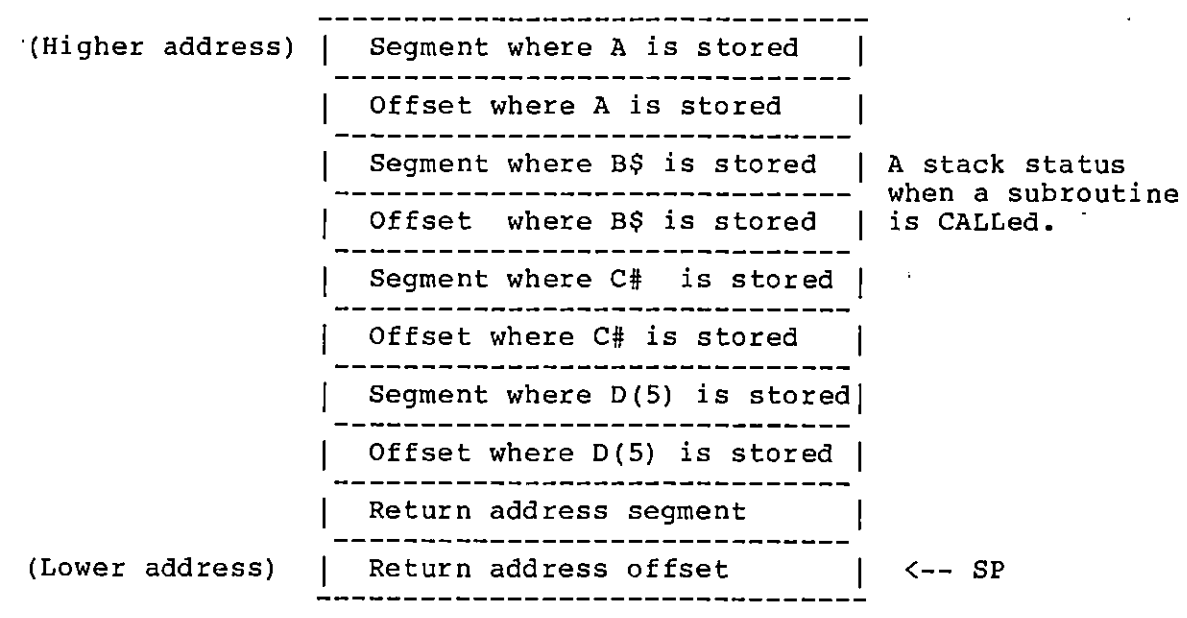
(1) <Start address> is the beginning address of an assembly language subroutine written with a constant or variable. Use a segment defined by the DEF SEG instruction.

(2) Any numberof variables can be specified as an argument. The arguments are passed to the assembler via a stack and their stored segment offsets are pushed sequentially.



(3) The return address is stored in the head of the stack with segment/offset order, because the control is given to the subroutine from the interpreter using 8086 LONG CALL.

EXAMPLE: CALL &HA000,A,B$,C#,D(5)


(Higher address)
Segment where A is stored
Offset where A is stored
Segment where B$ is stored
Offset where B$ is stored
Segment where C# is stored
Offset where C# is stored
Segment where D(5) is stored
Offset where D(5) is stored
Return address segment
(Lower address)
Return address offset
<-- SP
A stack status when a subroutine is CALLed.

(4) LONG RET after returning a stack to its original status by specifying (number of parameters x 4) as n in RETn to return from the subroutine.

Values of segment registers (DS,ES, and SS) at RET must be same as values when the subroutine was CALLed.

Write the start address using numeric expression and parameter using variable. Refer to an 8086 Assembly Language Programming Manual for assembly language.

```
10 MCSTAT=&HA000
20 CLEAR MCSTAT
30 READ A%,B!,C#
40 FOR ADRS=MCSTAT TO MCSTAT+36
50 READ CODE
60 POKE ADRS,CODE
70 NEXT ADRS
80 CALL MCSTAT,A%,B!,C#
90 PRINT A%,B!,C#
100 END
110 DATA 1,-3,5
120 DATA &H8B, &HEC           :'MOV BP,SP
130 DATA &H8B, &H5E, &H04     :'MOV BX,04[BP]
135 DATA &H8E, &H5E, &H06     :'MOV DS,06[BP]
140 DATA &H83, &HC3, &H07     :'ADD BX,7
150 DATA &H80, &H37, &H80     :'XOR BYTE [BX],80
160 DATA &H8B, &H5E, &H08     :'MOV BX,08[BP]
165 DATA &H8E, &H5E, &HOA     :'MOV DS,OA[BP]
170 DATA &H83, &HC3, &H03     :'ADD BX,3
180 DATA &H80, &H37, &H80     :'XOR BYTE[BX],80
190 DATA &H8B, &H5E, &HOC     :'MOV BX,OC[BP]
195 DATA &H8E, &H5E, &HOE     :'MOV DS,OE[BP]
200 DATA &HF7, &H1F           :'NEG WORD [BX]
210 DATA &HCA, &HOC, &HOO     :'RET 12
```

CHAIN/COMMON

Purpose

The CHAIN command causes the following to occur:

1. Enter from the disk to memory another program, which differs from the program currently stored in memory.

2. Transfer the contents and variables of the program in process to the newly entered program.

3. Execute the newly entered program.

With the CHAIN, a main program can be divided into sub-programs and continuously executed.

Form

```
CHAIN [MERGE] <file name> [,[<line number 1>] [,ALL]
   [, DELETE <line number 2> - <line number 3>]]
```

Description

(1) <File name> in this case is name of new program to be enterd.

(2) <Line Number 1> is the starting line of the newly entered program, from which the execution of the new program begins when specified. If <Line Number 1> is omitted, the execution starts from the first line of a subprogram.

(3) The variables of the currently executed program (preceding program) are transferred to the newly entered program (succeeding program) in the following way:

1. When all the variables of the preceding program are to be transferred to the succeeding one, the ALL option is specified by use of the CHAIN.

2. The variables of the preceding program are to be partially transferred to the succeeding one, ALL is omitted in the CHAIN statement, and only the necessary variables are specified by the COMMON statement.

Form

```
COMMON <variable> [( )] {,<variable> [( )]}
```

<Variable> in this case is to be transferred to the succeeding program. When it is an array, specification is written with enclosures as "A()".

The COMMON statement is effective regardless of its place in a program before the CHAIN.

If the COMMON statement has some errors, the error message

will be displayed when executing the CHAIN statement.

3. When the CHAIN statement in which the ALL option is specified is written for a program in which the COMMON statement is included, it is erroneous. The COMMON statement is prohibited when the CHAIN statement with ALL option exists in a program.

When the MERGE is specified, the execution is performed merging the preceding program with the succeeding one.

An overlay of subroutines becomes possible with this MERGE option. It is necessary to specify which part of the preceding program should be replaced by the succeeding program. The DELETE option is effective for this specification. In general format, the <Line Number 2> through to <Line Number 3> are deleted and replaced by the succeeding program. The DELETE option should be specified together with the MERGE option.

The CHAIN statement does not transfer the types of variables, therefore, the types of variables to be used in both programs should agree with each other beforehand.

<Line Number 1> is not effected by RENUM.

Examples

1)	CHAIN "PRGM-A"	A new program "PRGM-A" is loaded and executed instead of the currently executed program.
2)	CHAIN "PRGM-A", 200, ALL	"PRGM-A" is loaded instead of the currently executed program. All the variables in the preceding program are transferred to PRGM-A, and the execution of PRGM-A starts from the Line Number 200.
3)	CHAIN MERGE "SUB-A",DELETE 1000-2000	The part of the current program between the Line Numbers 1000 and 2000 is removed to merge in new program - SUB-A. The new program is created. Control

starts the new program

4) Program ("PRG!.BAS") which CHAINs.

```
100 ' filename =PRG1
110 Print "PRG1 START"
120 '
130 DIM A(12,5)
140 FOR I=0 TO 12
150 FOR J=0 TO 5
160 A(I,J)=J
170 NEXT: NEXT
180 B#=123456789
190 C$="ABCD"
200 '
210 CHAIN "PRG2"
220 '
230 COMMON C$, A()
```

```
Ready
>LOAD "PRG1"

Ready
>RUN
PRG1 START
PRG2 START
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5

0         ABCD
PRG2 END
```

Program ("PRG".BAS") which is to be CHAINed.

```
100 ' filename=PRG2
110 PRINT "PRG2 START"
120 '
130 FOR I=0 TO 12
140 FOR J=0 TO 5
150 PRINT A(I,J);
160 NEXT
170 PRINT: NEXT
180 '
190 PRINT
200 PRINT B#, C$
210 PRINT "PRG2 END"
220 END
```

"PRG1.BAS" execution

"PRG2.BAS" execution

CLEAR

Purpoes

This is an instruction to clear the variables of data. An optional function to reset the maximum address and the stack memory capacity of the SANYO-BASIC program is available.

Form

CLEAR [<P1>] [,<P2>]

Description

(1) When the CLEAR statement is executed, 0 is assigned to all the numeric variables and a null string to the string variables.

(2) When <p1> is specified, the maximum address in a SANYO-BASIC program can be reset. Originally, the maximum address is provided at the time in initialization but can be reset by use of this statement.

(3) When <p2> is specified, the stack memory capacity allocated in the SANYO-BASIC, 512 bytes are reserved for the stack memory. This memory area size, however, can be allocated again by the <p2> specification to tighten the loop nesting.) The stack memory capacity should be more than 255 bytes.

(4) An arithmetic expression may be used to write <p1> and <p2>.

Examples

1)
```
10 A = 34
20 B = 56
30 PRINT A + B
40 CLEAR
50 PRINT A + B
>RUN
 90
 0
```

2)
```
- CLEAR
- CLEAR &HCFFF
- CLEAR &HCFFF, 1024
```

CLOSE

Purpose

The CLOSE statement closes the disk file in use.

Form

CLOSE [[#] <file number> {,[#] <file number> }]

Description

(1) <File number> indicates the file currently opened and is specified with an arithmetic expression.

(2) When the specification of <file number> is omitted, all the open files are closed.

(3) The CLOSE statement releases the relationship between the particular disk file and its file number and uses the same file number to open a new file.

(4) The sequential output file completes an entry of data from the buffer into the disk file by the execution of the CLOSE statement.

(5) When the END or NEW statement is executed, all the open files are automatically closed even though no CLOSE statement is specified. The STOP statement does not close the files.

Examples

1)

```
  '
  '
>100 OPEN "R",1,"FILE.NAME"
  '
  '
>190 CLOSE #1  (Files opened under #1 are closed.)
>200 CLOSE     (All files are closed.)
  '
  '
>250 END
```

2)

```
100  OPEN "R", #1, "RND.DAT"
110  FIELD #1, 20 AS A$
120  OPEN "O", #2, "SEQ.DAT"
130  '
140  LSET A$="BASIC"
150  PUT #1, 1
160  PRINT #2, "ABCDE"
170  CLOSE #2
180  '
190  OPEN "I", #3, "SEQ.DAT"
200  INPUT #3, B$
210  GET #1, 1
220  PRINT A$, B$
230  CLOSE

>RUN
BASIC                ABCDE
```

CLS

Purpose

The CLS statement erases out the images of the cathode-ray tube display.

Description

(1) When the CLS statement is specified, the visual images are all erased. The image attributes are then reset. The Cursor appears at the position of Line 1 and Row 1 (the upper-left side of the screen).

Example

```
>10 REM CLS STATEMENT TEST
>20 X$ = "PG. 2":Y$ = "PG. 1"
>30 PRINT Y$
>40 INPUT "CONTINUE? (Y OR N)";Z$
>50 IF Z$ = "N" THEN 50
>60 CLS
>70 PRINT X$
>RUN
```

CONT

Purpose

The CONT resumes a program which has been halted.

Form

CONT [N]

Description

(1) The program, which has been entered into the stopped state by use of the CTRL + C is resumed.

(2) If any modification is made during the stopped state, the CONT becomes invalid.

(3) CONT [N] has the following functions in addition to the above functions:

a. Prohibites suspension of program execution by the CTRL + C

(4) When CONT is entered again in the CONT N Mode, the above functions (a, b, c) are cancelled.

Example

```
10  PRINT "'START'"
20  PRINT "Please 'CONT'"
30  STOP
40  '
50 PRINT "'END'"

>RUN
 'START'
Please 'CONT'

Break in 30

Ready

>CONT
 'END'
```

CREATE

Purpose

CREATE allocates index sequential file area on a disk.

Form

CREATE <filename>, <record length>, <key position>, <key length>,<number of records>

Description

1) <filename> is the name of the file to be allocated.

2) <record length> is the number of bytes of one record of the file - between 2 and 256.

3) <key position> specifies which byte is to be the key at the beginning of the field.

4) <key length> is the length of the field which is to be the key - 1 through 30 bytes.

5) <number of records> can be up to 32,767.

6) CREATE is valid only for index sequential files. If enought space is unavialble on a disk, the area cannot be allocated. Area may be calculated as follows:

Data part

$$\frac{\text{record length x number of records}}{128} = \text{DA (rounding)}$$

Key part

If key length is 1 to 14, KL=16, BC=63
If key length is 15 to 30, KL=32, BC=31

$$\frac{\text{number of records x 2}}{BC} = \text{KB1 (rounding)}$$

$$\frac{\text{KB1 x 2}}{BC} = \text{KB2 (rounding)}$$

$$\frac{\text{KB2 x KL}}{128} = \text{KB3 (rounding)}$$

$$\frac{\text{DA+KB3+1}}{8} + \text{KB1} + \text{KB2}$$

= file K bytes (area) (rounding)

7) A character expression can be used for the filename while a numeric expression is used for the record length, key position, key length and number of records.

DEF FN

Purpose

The DEF FN statement defines the type of user functions to be specified in a line.

Form

DEF FN <name> [(<parameter> {,<parameter>})] = <expression>

Description

(1) <Name> is to call the function defined by the DEF FN statement based on the same rules as a variable name. (An alphanumeric string beginning with a letter with the first two characters identifying the type.) The format "FN <name>" is used to call it.

(2) <Parameter> may be omitted. (A function with no parameter can be defined.) When it is not omitted, the parameter is assigned with the variables in <expression>. Then it is replaced by the parameter (actual parameter when the function is called out) to evaluate the value of expression.

(3) <Expression> defines the evaluation method of the function and can be included in a DEF FN statement line of up to 255 digits.

The variables in the expression simply define the evaluation method. These are "dummy" variables and have no effect on the variables in a program even if they happen to be the same.

A function can be called using this format:

FN <name. [(<actual argrument>{,<actual argument>})]

All the variables in the expression do not necessarily become the parameters of the statement. When a certain variable is used for the parameter, it can be regarded as a real parameter in the calculation, and when not used, the real value of the variable will be calculated.

(4) The value of the calculated function is converted to the type code. If the conversion does not work, an error will occur at the line where the function is quoted.

(5) The DEF FN statement should be defined before its reference. It is not used in the Direct Mode. Any undefined user function must not be used as an item of the expression. An expression can be used for actual argument.

Example

```
>10 DEF FNA (X) = 3.14 * 2 * X + B ^ 5
>20 B = 5.0
```

```
>30 C = FNA(25.0)  (C is calculated by
                    3.14x2x25+5^5.)
>40 PRINT FNA(X)
>RUN
  182
```

DEFINT/DEFSNG/DEFDBL/DEFSTR

Purpose

These statements define the variable types (integer, single-precision, etc.) by their first and succeeding letters.

Form

DEF<type> <Character 1> [-<Character 2>]{,<Character 1> [-<Character 2>]}

Description

(1) Either INT, SNG, DBL, or STR is assigned to <type>.

(2) <Character 1> is the first letter and <character 2> indicates one of the remaining letters which follow the first character. The type of variable name beginning with the letter of <Character 1> through <Character 2> takes on the <type> specification.

(3)The type specification codes are interpreted prior to the declaration. For the variable with no type specification code or DEF specification, a single-precision real number is assumed.

(4) The type specification with a DEF declaration can be defined again in the midst of the program.

(5) No spaces exist between DEF and <type>.

Example

```
>10 DEFINT I-N  (The variables beginning with
                 I, J, K, L, M or N will be
                 integers.)
>20 I = 34.17:J = 23.78
>30 PRINT I + J
>RUN
58
```

DEF SEG

Purpose

DEF SEG sets a segment base address when specifying memory address using POKE, PEEK, CALL , or USER.

Form

DEF SEG [=<address>]

Description

(1)<address>isan expression with avaluebetween0and &HFFFF used as an actual address with an offset address specified by each instruction.

(2)The Segment base address initial value is the segment address of the data segment of the interpreter. If <address> is omitted, it is the initial value. Therefore, it VARPTR function value (refer to Chapter 10) is used as an offset address (if operating a user program area), use the initial value as the segment address.

DEF USR

Purpose

The DEF USR statement defines a subroutine written in assembly language. The CALL statement can write with up to 3 parameters. The DEF USR provides only one parameter and transfers a single execution result from the subroutine program to the main program. This is the important difference between the two statements.

Form

DEF USR [<n>] (variable) = <start address>

Description

(1) Only one number between 0 and 9 is assigned to <n>. When omitted, the default value 0 is assumed. Ten types of user functions can be defined in a SANYO-BASIC program using an Assembly language subroutine.

(2) (variable) only defines function type.

(3) <start address> is an offset of a subroutine start address and can be written using an ex@ression. A segment address defined using DEF SEG is used.

(4) The user function can be called using this format:

USR [<n>] (<actual arguments>)

In this case <n> is the same number specified in the DEF USR statement.

<actual Argument> is the parameter passed and can be written using an expression. Its result is converted to <varialbe> type. If the conversion is impossible, an error will be indicated.

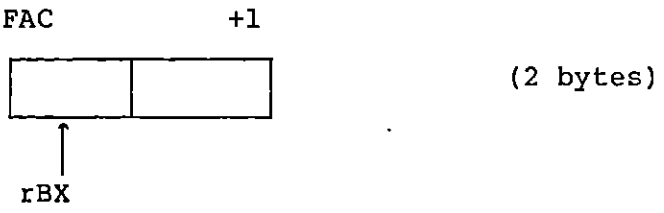
(5) The parameters are transferred to the subroutine with the following procedure:

A. The data type code is stored in an AL register

Type of Parameter	Code (AL Register)
Integer	2
Character string	3
Single precision real number	4
Double-precision real number	8

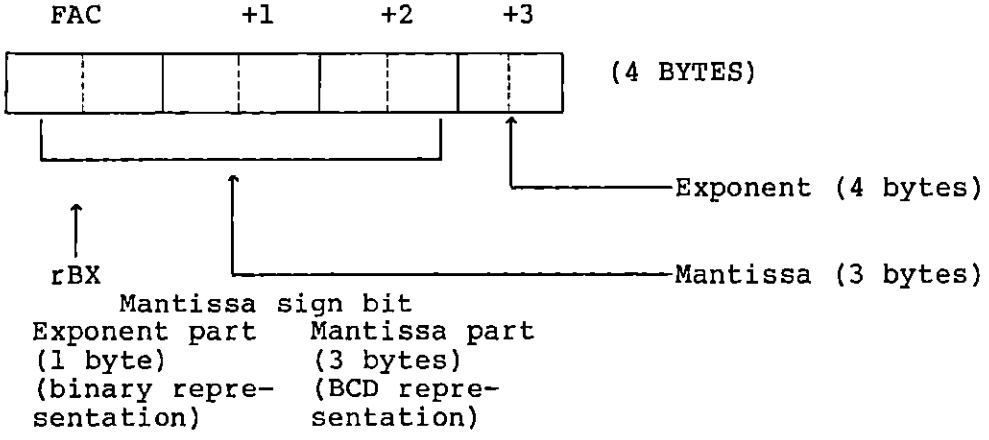
B. When the parameter is a numeral, it is loaded in the floating point accumulator (FAC) of memory. The FAC address is loaded in the HL register. The contents of FAC are clearly shown:

Integer (rAL=2)



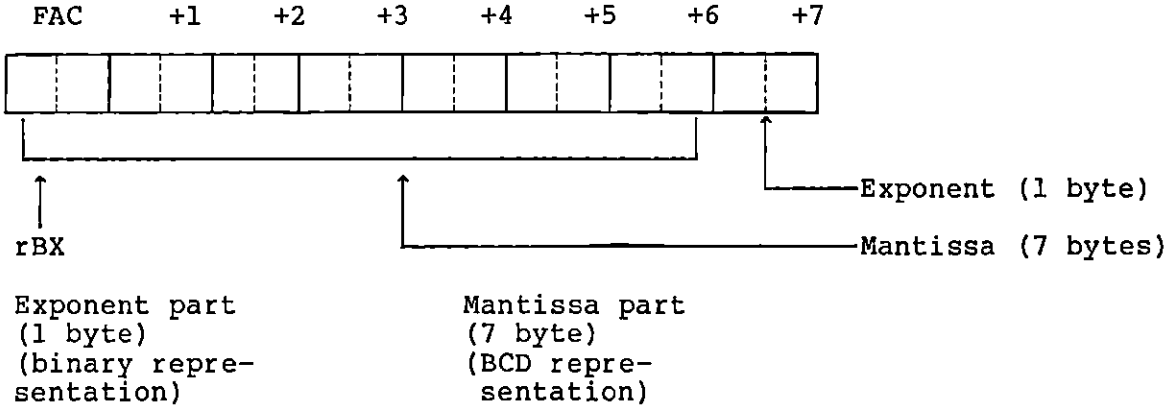
Integers are expressed using "two's" complement if negative. An integer is positve if the highest bit (7th bit) of FAC higher byte is 0; negative if the highest bit is 1.

Real number (rAL=4)



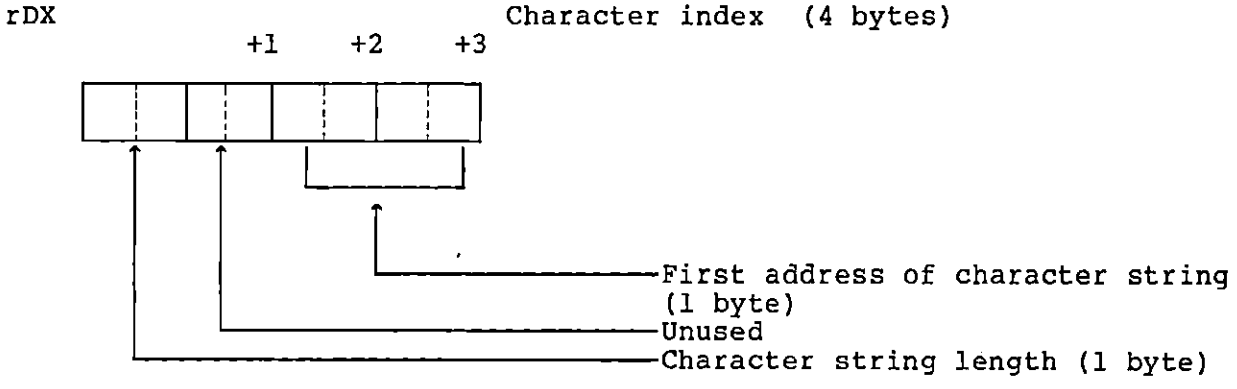
When a single precision real number is used, a 6 digit mantissa expressed using BCD is retained in an FAC lower 3 bytes. The entire sign is indicated by the highest bit (7th bit) of the higher 1 byte. indicates the real number is positive, 1 indicates negative. The exponent is expressed using the rest of the 7 bits (bits 0 through 6) and is retained by a binary number.

Double-Precision Real Number (rAL=8)



When a double precision real number is used, the mantissa is retained in the FAC lower 7 bytes. The sign and exponent are retained in the higher byte. The contents are the same as a single precision real number.1

C. When the parameter is a character string, the first address of the character index (4 bytes) is stored in the DX register.



The lowest 1 byte of the character index retains the character string length (0 to 255). The higher 2 bytes retain the first address offset.

The subroutine execution result must be the same as the parameter type when called. Store the result in the FAC or position where the parameter character string is stored if the function value is numeric. Then specify LONG RET. The character string execution result must be the same as the parameter length. The character string index canot be modified by the user. Recover the values of segment registers (DS, ES, and SS) and stack pointer (SP) to their states when the subroutine is called before returning from the subroutine.

Example

```
   :
   :
>100 A = &HA000
   :
   :
>150 DEF USR5(p)=A+&H1000
   :
   :
>200 X=USR5(Q)*2
   :
   :
>300 END
```

DELETE

Purpose

This instruction deletes a part of the program in the Program Edit Mode. After execution, control returns to the SANYO-BASIC Mode.

Form

DELETE [<Line Number 1>] [-<Line Number 2>]

Description

(1) The following combinations are available for the specification of line numbers:

a. <Line Number 1>-<Line Number 2>
   Deletes the part between the specified two lines

b. <Line number> - . (period)
   Deletes the part from the specified line number up to the end of the program.

c. -<Line number>
   Deletes the part from the beginning of the program up to the specified line number.

d. <Line number>
   Deletes the line with the specified line number only.

(2) When no program exists in the range specified for deletion an error occurs.

(3) The NEW statement is used for the entire deletion of the program.

(4) A period (.) indicates the final line of the program.

Example

```
>10 A% = 26 * 82
  '
  '
>90 PRINT A%
>100 B% = A% * 2
>110 PRINT B%
>DELETE 20-90
>DELETE 100-
```

DIM

Purpose

The DIM statement declares an array specification.

Form

DIM <variable>(<expression>{,<expression>){,<variable>(<expression>{,
<expression>})}

Description

(1) <Variable> indicates the name of the array.

(2) The value of <expression> determines the number of array elements in the corresponding dimension. And the number of <expression>, which follow <variable>, indicates the dimensional size of the array. No limitation is provided on the dimensional size or on the number of elements, however, both will be included in the total capacity of the memory.

(3) The DIM statement must be declared before the specified array is used. To get the evaluation values of each <expression>, they are rounded (counting the fractions five and over and disregarding the rest). When the result is a negative value, an error occurs.

(4) Under the array declaration, the elements are all cleared to 0 when the array is of numbers and replaced by null strings when it is of characters.

(5) The minimum value of a subscript: If declared as DIM A(10), the number of elements will be one of 11 values from 0 to 10. The OPTION BASE statement can modify the minimum value to be 1.

(6) All arrays should be declared before use. When the maximum value of the subscript is less than 10, the array can be used without declaratin. Regardless of the dimensional size, 10 declarations are assumed in each dimension when the array is used for the first time.

(7) The array declaration can be cancelled by the ERASE statement .
(When the total capacity of memory is filled up, the user can erase unnecessary arrays, then make a declaration of new arrays.)

Examples

1)
```
>10 DIM A(10,20,10)
>20 FOR I = 1 TO 10
>30 FOR J = 1 TO 20
>40 FOR K = 1 TO 10
>50 INPUT A(I,J,K)
>60 NEXT K, J, I
  '
  '
```

2)

```
>10 INPUT M, N
>20 DIM A(M, N)
>30 FOR I = 1 TO M
>40 FOR J = 1 TO N
>50 INPUT A(I, J)
>60 NEXT J, I
   '
   '
>200 END
```

EDIT

Purpose

To edit a line of a program, type EDIT then the line number. The line appears on the screen with the cursor one line below.

Form

EDIT <LINE>

Description

(1)The DELETE key is pressed to delete characters. A "delete" character shows the position of deleted characters.

(2) The <-- cursor left key erases the edited line and puts the prompt at the beginning again. All characters entered are inserted into the old line at the point where the prompt character is. The RETURN key leaves the EDIT mode and enters the new line as changed.

(3)The BREAK key leaves the EDIT mode and the line unchanged.

END

Purpose

The END statement terminates the execution of the program, closes all the opened files and returns control to the BASIC Mode.

Description

(1) The user can write the END statement anywhere in the program. If there is an open file at the time when the END statement is specified, it will be closed. The control stops the program operation and returns to the BASIC Mode.

(2)When no END statement is found at the end of the program, the system performs the same operations automatically (closeing all files and going into the BASIC Mode).

Example

```
500 END
```

ERASE

Purpose

The ERASE command erases an array from the program and makes the corresponding memory area usable for another purpose.

Form

ERASE <array name> {,<array name>}

Description

(1) When the array is erased, the corresponding memory area becomes blank. Using the same or another array name, a new array can be declared.

(2) The ERASE command cannot be used in the FOR/NEXT loop, WHILE/WEND loop, or subroutines.

Example

```
>10  DIM A$(100)
  '
  '
>150 ERASE A$
  '
  '
>300 END
```

ERROR

Purpose

The ERROR has two functions. One is the error simulation to intenionally produce the errors defined in SANYO-BASIC, the other is a user defining function to specify errors of the user's own choice.

Form

ERROR <arithmetic expression>

Description

(1) When the evaluation value of <arithmetic expression> agrees with the error number already defined in SANYO-BASIC, the corresponding error occurs. (See SANYO-BASIC error massage list in the technical reference section.)

(2) When the value is greater then the error number of SANYO-BASIC definitions, it works for the user error defining function. The error number defined by the user can be used in the execution of the error trap subroutine.

(3) The value of <arithmetic expression> should be an integer in the range between 1 and 255. When it falls out of the range, an error occurs. When it is a real number within the range, it is rounded (counting the fractions 5 and over as a unit and disregarding the rest).

(4) When an error message is entered with an error number which does not exist in the list, an error occurs.

(5) When an error occurs or is produced intentionally, the system displays the error message, stops the execution of the program and returns the control to the SANYO-BASIC Mode.

(6) When an error occurs, the ON ERROR GOTO statement works for:

Stopping the control from returning to the SANYO-BASIC Mode
Performing error processing, and
Generating the error trap subroutine to resume the program.

Example

```
 '
110  ON ERROR GOTO 500
 '
 '
150  IF A>100 THEN ERROR 200
 '
 '
490  END
500  IF ERR = 200 THEN 600
 '
 '
600  (Action when A>100)
650  RESUME NEXT
```

FIELD

Purpose

The FIELD assigns a field name and its length in the buffer area for use of random files.

Form

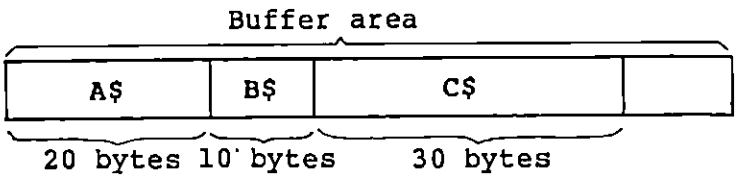
```
FIELD [#] <file number>, <field length> AS <string variable>
                  {,<field length> AS <character variable>}
```

Description

(1) <File number> specifies the particular file which has been opened. It is written with <arithmetic expression>.

(2) <Field length> indicates the length (number of bytes) to be allocated to <string variable>.

```
>100 FIELD #1, 20 AS A$, 10 AS B$, 30 AS C$
```



(3) The FIELD statement, as shown above, assigns a name to each field and determines the field length. The LAST or RSET statement should follow the FIELD statement to enter the data of disk file into the buffer area, the the PUT statement performs the write operation into the disk file.

(4) The total of field lengths specified in the FIELD statement must not exceed the record length of a random file defined by the OPEN statement.

When the user would like to modify the name and field lengths of a buffer area:

1. Resets the program by keying in as follows:

   ```
   >200 FIELD #1, 0 AS A$, 0 AS B$, 0 AS C$
   ```

2. Enters a new FIELD statement for reassignment.

The string variable defined in the FIELD statement should not be redefined in the INPUT, LET, READ, SWAP, OR MID# statement.
An array element is prohibited for use.

FILES/LFILES

Purpose

The system sends the file name stored in the disk to the display or the printer. FILES is for the display and LFILES for the printer.

Form

[L] FILES [<file name>]

Description

(1) When <file name> is omitted, all the file names of the disk which are currently installed at Drive A will be output.

(2) When a question mark (?) is included in the file name it becomes a substitute for any of the characters. So, the system displays or prints out all possible characters other than ?.

(3) An asterisk (*) can be used to replace all the file names or extended file names.

Example

```
>FILES            (Indicates all files in Unit A.)
>FILES "B: "      (Indicates all files in Unit B.)
>FILES "*.BAS"    (Indicates all files which have
                   extension BAS.)
>FILES "A????.*" (Indicates all files whose name
                   is of 5 digits beginning with A.)
```

FOR/NEXT

Purpose

This instructs the system to loop as many times as specified. The FOR and NEXT statements must be used together.

Form

FOR <variable> = <arithmetic expression 1> TO <arithmetic expression 2> [STEP <arithmetic expression 3>]

NEXT [variable> {,<variable>}]

Description

(1) <Variable> indicates the number of repetitions to be performed.

(2) <Expression arithmetic 1> calculates the initialization value to count the number of repetitions. <Arithmetic expression 2> produces the end count value. the STEP value <arithmetic expression 3> is used to increment (positive number) or decrement (negative number) the counter with each loop. When the counter value does not reach the termination value, the control returns to the FOR statement and repeats until the counter value reaches the termination value. When reaching the end value, the control jumps to the next statement after NEXT.

(2) If the counter's intitialization value exceeds the termination value, the loop will not be executed. The control jumps to the statement written after the NEXT statement. When more than one NEXT statement is described for a single FOR statement, the control jumps to the statement which follows the NEXT statement with the smaller line number (nested loops).

(3) When STEP declaration is omitted, the default value "1" is assigned for <expression arithmetic 3>. The counter is incremented by 1 each time.

(4)The FOR and NEXT loop can be nested. When nested, the variables for the counter value should be defined in each of the loops.

(5)If the termination line is common to the nested loops, the variables can be written in one NEXT statement. However, the counter variables for the inner most loop should be written on the left side of the outer most loop's variable.

(6) When the variables are omitted in the NEXT declaration, they are assumed the same as for the preceding FOR statement. The variables which specify the termination line (NEXT) common to nested loops cannot be omitted.

(7) A double-precision real number cannot be assigned to <variable> (the variable for repetition control of a loop).

(8) If the procedure to exit from the FOR/NEXT loop without ending by a NEXT is repeated, an error message will appear.

Examples

1.

```
>10 FOR I = 1 TO 10
>20 INPUT A
>30 NEXT I
```

```
>10 FOR A = 1 TO 10 STEP 0.5
>20 FOR B = 1 TO 20
>30 FOR C = 1 TO -10 STEP -1
  '
  '
>200 NEXT C, B, A
  '
  '
>300 END
```

2.

```
>10 INPUT M, N
  '
  '
>100 FOR I = M TO M + N
>110 INPUT S
>120 NEXT I
  '
  '
>300 END
```

GET

Purpose

The GET loads the data from the random access file to the random buffer which has been defined by use of the FIELD statement.

Form

GET [#] <file number> [,<record number>]

Description

(1) <File number> is the number under which the file was opened for "R" mode.

(2) <Record number> is the file record number in the range between 1 and 32767. When it is omitted, the next record after the last GET or PUT is assumed.

(3) Both <file number> and <record number> may be written with <arithmetic expression>.

(4) If a record number which has not been PUT by a user is then used in a GET, the content of the read-out record will not be correct.

(5) Index sequential file has this form:

GET [#] <file number> [, <key>]

A) <file number is the number used to open the file to be read. It must be opened with 'IS' mode. Numeric expression is used fot the file number.

B) <key> is the key of the data to be read. Use character expression. If a key of the data which was not PUT is specified, an error is indicated. If this is omitted, the next record pointer of PUT, GET, SETKEY, or KILL executed immediately before is assumed. Sequential read is possible using this method.

Example

```
  '
>150 GET #1
  '
>200 GET #2, 501
  '
>250 GET #N, REC%
  '
  '
>400 END
```

GOSUB/RETURN

Purpose

The GOSUB or RETURN instructs a branch from the main program to the subroutine program or return from the subroutine to the main program.

Form

```
GOSUB <line number>
---
RETURN [<line number>]
```

Description

(1) <Line number> is the starting line number of the subroutine program. <Line number> must be <integer constant>.

(2) When RETURN is executed the control returns to the next statement after the GOSUB. when <line number> is specified, the control goes to the specified line number.

(3) Calls of subroutines can be nested. (Another subroutine program can be called by a subroutine program.) However, the currently executing subprogram cannot call itself.

(4) Following a GOTO, STOP, or END statement, a subroutine program may be entered in any part of the main program. Otherwise, the main control might accidentally fall into the subroutine program.

Example

```
   '
   '
>300 GOSUB 500
   '
   '
>490 GO TO 540
>500 REM GOSUB ROUTINE #1
>510 FOR I = 1 TO 10
>520 INPUT A$
>530 NEXT I
>540 END
```

```
   '
   '
>200 GOSUB 1000
   '
   '
>900 END
>1000 REM GOSUB ROUTINE #2
>1010 FOR I = 1 TO 10
>1020 INPUT A$
>1030 NEXT I
>1040 RETURN
```

GOTO

Purpose

This is an unconditional branch instruction.

Form

GOTO <line number>

Description

(1) <line number> is the integer constant indicating the destination (line number of the branch).

(2) When the GOTO statement is executed, the control goes to the specified line number.

(3) If the specified line number is not found in the program, an error occurs.

Example

```
>10 INPUT A; B
>20 C = A + B
>30 IF C < 1000 GOTO 100
>40 PRINT "A + B IS GREATER THAN 1000"
  '
  '
>100 END
```

HCOPY

Purpose

HCOPY directs the entire screen as it is to the printer.

Description

(1) Only standard printers can be used.

(2) If a key is pressed during the HCOPY execution, processing is terminated.

IF/THEN/ELSE

Purpose

The IF statement is a conditional diverging instruction by which the control of the program may be diverged under certain circumstances.

Form

```
IF <expression> THEN <statement> {:<statement>} <line number>
                [ELSE <statement> {:<statement>} <line number>]
or

IF <expression> GOTO <line number>
                [ELSE <statement> {:<statement>} <line number>]
```

Description

(1) <Expression> indicates the condition for diverging. When the evaluation value of <expression> is not 0 (true), the control executes the statement after the THEN or the GOTO statement. When it is 0 (false), the ELSE clause is executed. If ELSE clause is omitted, and when <expression> results are (false) 0, the control jumps to the statement after the IF statement. When <expression> does not result in 0 and there is no GOTO or THEN clause, the control jumps to the statement after the <expression>.

(2) An IF statement can be written in the part after a THEN or ELSE. (This means that the IF instructions can be nested.) There is no limitation on the depth of a nest; however, the total statement can only be up to 255 characters long.

(3) When the IF statement are nested, it is understood that an ELSE statement will be included in the IF statement immediately before. The ELSE statement cannot be excluded from a loop of IF statements. It can only be omitted in the outermost IF statement.

Example

```
>10 INPUT A
>20 D = 10: H = 25
>30 IF A>3 THEN A = A-1: C = D: H = H+2: GOTO 200 ELSE 30
  '
  '
>200 PRINT A, C, D, H
>210 END
>300 PRINT A, C, D, H
>310 END
```

INIT%

Purpose

The INIT% statement sets the sending/receiving transmission parameters of the RS-232C serial port.

Form

INIT% <port number>, <expression 1>, <expression 2>

Description

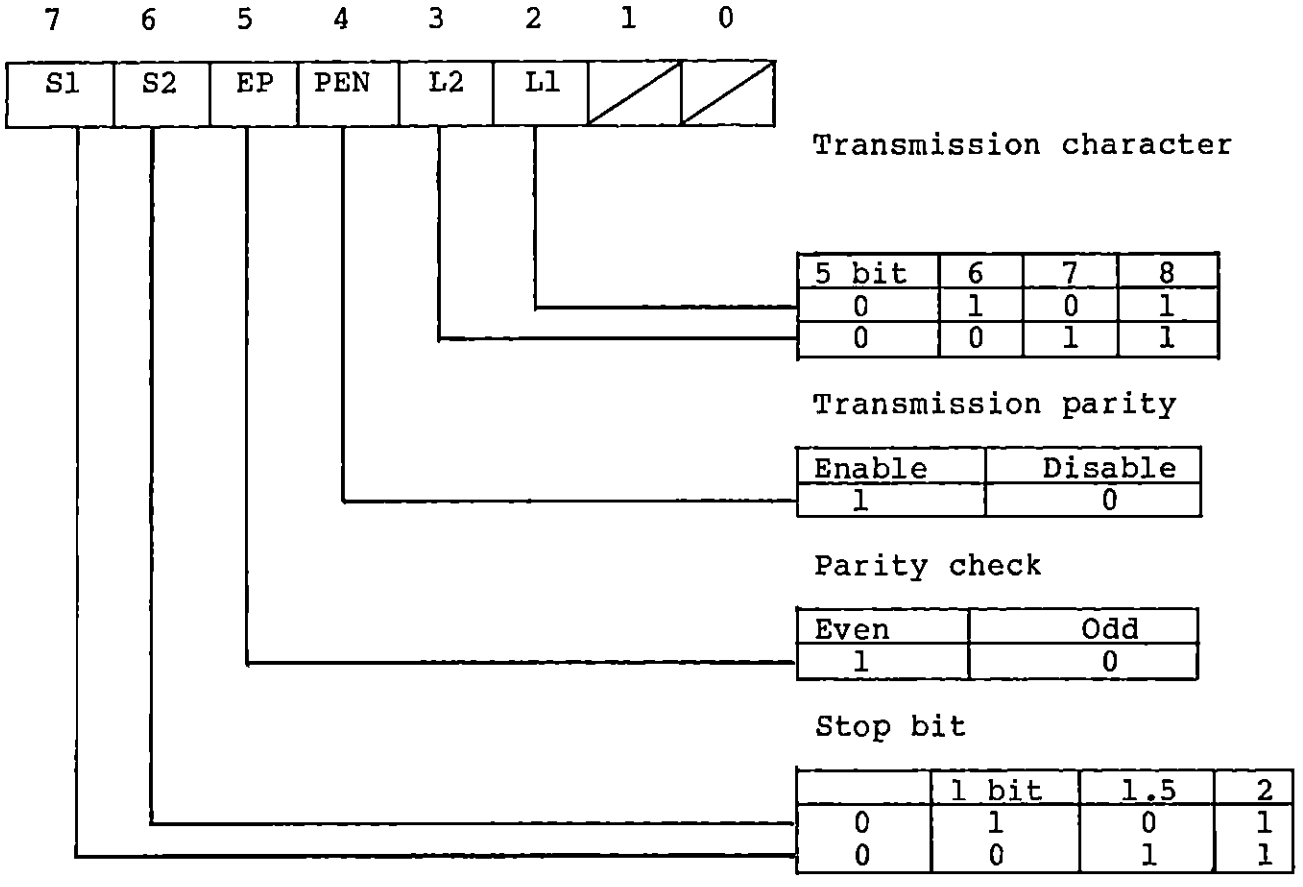
(1) <port number> takes only 1 and corresponds to the line connector socket in the rear panel.

(2) <expression 1> sets the parameter of 1 byte for sending/receiving transmission.

(3) <expression 2> sets the 1-byte parameter as data for specifyingsending and receiving transmission speed.

Example

<Expression 1> = mode setting (1 byte)



<expression 2> = transmission speed setting (1 byte)

Evaluation value of Expression 2	0	1	2	3	4	5	6	7
Baud rate	9,600	4,800	2,400	1,200	600	300	150	110

If the INIT% statement is not executed, the following initialization definition is made for the system. Characters: 7 bits, parity = enabled, even number parity check, stop bit: 1; 9,600 bps.

(<Expression 1> = 120, <Expression 2> = 1)

Untility LINE CMD enables this function to be performed under CP/M-86.

INPUT

Purpose

This is an instruction to enter data through the keyboard.

Form

INPUT [(<number of digits>)] [;] [<message>;|,] <variable> {, <variable>}

Description

(1) When the control comes from the INPUT statement, the system stops the program processing and waits for data from the keyboard. When the required data is entered with <CR> at the end, it is stored as a variable and the program goes to the next step.

(2) The number of digits to be entered is the same as the number of variables written in the INPUT statement. An integer 1 through 255 can be specified; the fraction part is automatically rounded.

(3) When <message> is specified, the message appears on the screen while waiting for keying. This message notifies the operator the data to enter. If message is omitted, a question mark (?) is displayed. Character expression is used for entering the message.

(4) The semicolon (;) before message is the control of <CR> code after the data is entered. When the semicolon is written, the Cursor stays immediately after the keyed-in data item as <CR> is not sent to the display. Another data item will be displayed following the previous one. When the semicolon is omitted, <CR> to indicate the termination of data entries is sent to the display, and the Cursor is returned and fed to the next line. characters on the next line are deleted.

Select either ; or , after message. If ; is selected, "?" is displayed after the message. It is not displayed if , is selected.

(5) The operator must enter as many <variable> data as written in the INPUT statement. Separate the data corresponding to each <variable> using a comma (,). If the entered data type differs from the <variable> type or the number of entered data differs from the number of <variables>, "Redo from start" is displayed and the system waits for keying.

(6) A comma (,) cannot be included in a character string to be entered when keying a character data. A comma is assumed as a delimiter. Use LINE INPUT to enter the character string having commas.

Examples

(1)

```
>10 INPUT "PLEASE KEY-IN A, B, AND C!"; A, B, C
>20 D = (A + B) * C
>30 PRINT D
>40 END
```

(2)

```
>10 INPUT (15)X
```

INPUT#

Purpose

The INPUT# reads data from a sequential file and stores it in the specified variable.

Form

INPUT# <file number> <variable>{,<variable>}

Description

(1) <File number> is the number under which the file was opened in "I" mode.

(2) <Variable> stores the data read from the file. The type of variable should agree with that of the data.

(3) The system reads the data skipping the space and <LF> in the beginning.

(4) When the character data begins with quotation marks (") the data is read in as one item until the next set of quotation marks. A set of quotation marks cannot be included between a set of quotation marks.

(5) One data item continues until a comma (,), CR, LF, or the 255th character comes. If the user wishes to use comma(s) in the data item, the LINE INPUT# is available.

Example

```
  '
  '
>50 OPEN "I", #3, "FILE-A3"
  '
  '
>100 INPUT #3, A$, B$
>110 CLOSE
  '
  '
>250 END
```

INPUT%

Purpose

The INPUT% reads data from the RS232C serial port and stores it in the specified variable.

Form

INPUT% <port number>,<character variable> {,<character variable>}

Description

(1) <port number> takes only 1; it corresponds to the line connector socket in the rear panel.

(2) <Character variable> stores the data entered from the port. The transmitted data are all processed as character data.

(3) A comma (,), CR code, CRLF code, or 255th character separates the entered data. The data following one of them will be stored in the next variable area.

(4) For the entry of data in which a comma is included, the LINE INPUT% statement can be used.

Example

```
  '
  '
>50 INPUT% 1, &HFE, 3
  '
  '
>100 INPUT% 1, A$, B$, C$
  '
  '
>200 END
```

NOTE: To terminate the wait loop caused by INPUT%, the sending device must send a carriage return. When the carriage return is received, the character variable will contain the characters received.

KEY

Purpose

The KEY assigns character strings to each of the function keys (PF1 TO PF15) When several character strings have been assigned to PF keys in advance, the system performs the same operations as these character strings by pressing only the necessary PF keys.

Form

KEY <PF key number>, <string expression>

Description

(1) <PF key number> indicates the number of a PF key for which a certain string is to be assigned. It is written with <arithmetic expression>. The specified key number falls under the range between 1-15.

(2) <String expression> consists of characters with up to 15 digits assigned to the respective function keys. When it exceeds the limit, the remaining part of the string will be cut off.

Example

```
>10 KEY 14, "RUN" + CHR$(&HOD)
   (The RUN  <CR> function is assigned to PF14.)
>20 KEY 1, "TAB"
  '
  '
>200 END
```

KEY LIST/KEY LLIST

Purpose

The KEY LIST statement works for listing the contents of the data which is assigned to the function keys on the screen. The KEY LLIST statement sends the list to the printer.

Form

KEY [L]LIST

Description

(1) With the KEY LIST statement, the control codes like CR, LF, etc. are also displayed on the screen. When printing out with the KEY LLIST statement, the control codes are converted to question marks (?).

Example

```
> KEY LIST
```

KEY LOAD/KEY SAVE

Purpose

The KEY SAVE statement functions to store the assignment list onto a disk file, and the KEY LOAD statement reads the stored assignment list into the memory.

Form

```
KEY LOAD <file name>
KEY SAVE <file name>
```

Description

(1) <File name> is the name of file into which the assignment list of function keys is to be saved. Several kinds of assignment lists can be stored in the disk, with the most suitable assignment list loaded depending on the required operation.

Example

```
>KEY SAVE "B:K-TBL-1"
>KEY LOAD "B:K-TBL-1"
```

KILL

Purpose

[1] This instruction deletes the specified disk file.

Form

KILL <file name.xxx> {,<file name.xxx>}

Description

(1) <File name.xxx> specifies the particular file to be deleted. The file currently opened cannot be specified. The file name qualifier, .xxx must be specified.

(2) Every character can be substituted by a question mark (?) in specifying <file name.xxx>. An asterisk (*) can be also assigned to all file names or extended file names.

(3) The KILL instruction is valid for all the disk files and can be used within a program.

Example

```
>KILL "FILE-A.BAS", "FILE-B.CMD", "B:DATA-1.DAT"
>KILL "TEST??.*"
>KILL "B:*.*"
>KILL "C:*.DAT"
```

[2] KILL
The KILL instruction deletes the data having the specified key from the index sequential file.

KILL # <file-number>,<key>

- <file-name> is the number used to open the file to be deleted and can be specified using the numeric expression. The file must be opened with an "IS" mode.
- <key> is the key of the data to be deleted and can be specified using the character expression. An error ocurs if the key of the data are not PUT is specified.

LET

Purpose

The LET statement places the storage of data into the area of variables.

Form

[LET] <variable> = <expression>

Description

(1) The key word LET may be omitted.

(2) When <variable> is a numeric value and the expression is a character, an error occurs.

(3) When <variable> and <expression> are both numeric but their types are different, the operation of the <expression> comes first, then conversion of the evaluatin value into the type code follows for storage. In case the conversion cannot be performed, it is erroneous.

Example

```
>10 A = 1: B = 2: C = 3
>20 A = A + 1
>30 A$ = "ABC" + CHR$(&H41)
>40 B# = 0.123456789D-5
>50 C% = 231.5
>60 PRINT A, A$, B#, C%
  '
  '
>200 END
```

LINE INPUT

Purpose

The LINE INPUT statement is for storing the entered character strings in the character variable.

Form

LINE INPUT [(<number of digits >)] [;] [<message>;] <character string>

Description

(1) Similar to INPUT, the LINE INPUT reads the data of string type into <character variable>. <CR> should be written at the end of data.

(2) This statement, as the INPUT statement, reads the keying data into the character variable as a character string. The data is stored in the character variable when the RETURN key is pressed.

(3) <message> is intended for the operator and is displayed on the screen during data entry. "?" is not displayed if this is omitted.

(4) A semi colon (;) between the <number of digits> and <message> is the RETURN code control after the keying. If a colon (:) is specified, the cursor remains immediately after the keying data and the program procedes to the next step. If it is not specified, the cursor is carriaged-returned after the keying (Feed Line is deleted).

(5) The INPUT statement assumes the comma (,) as a data delimiter, but the LINE INPUT reads the comma (,) as data.

(6) <number of digits> is the number of digits of the keying data used to check and can be written using numeric expression. An integer 1 through 255 can be specified. The fraction part is automatically rounded. The number is indicated as an error if it exceeds the limit. When <number of digits> is specified, up to <number of digits> can be entered, although only the RETURN key can be entered after that. All other input is ignored.

Examples

1)
```
>10 LINE INPUT "address"; AD$
>20 LINE INPUT "data key-in"; X$
>30 LINE INPUT (25); "address"; A$
>40 LINE INPUT (A + B); Y$
  '
  '
>200 END
```

2)

```
>10 INPUT "NAME?",A$
>20 LINE INPUT "ADD.?",B$
>30 INPUT "TEL.?",C$
>40 PRINT A$; ":"; B$; ":"; C$

>RUN

NAME?  Ron Milos
ADD.?  51 Joseph Street
TEL.?  440-9300

Ron Milos:51 Joseph Street:440-9300
```

LINE INPUT#

Purpose

The LINE INPUT# statement has the same format as the LINE INPUT and reads the character data of the disk file (sequential file) into the specified variable.

Form

LINE INPUT# <file number>, <string variable>

Description

(1) <File number> is the number under which the file was opened in "I" mode.

(2) One data item to be read out of the specified sequential file into <string variable> continues from the beginning to <CR>. In the INPUT# specification, a comma (,) is regarded as a delimiter of data; however, in the LINE INPUT specification, the data is read out until <CR> appears including comma(s).

Examples

1)

```
  '
  '
>100 OPEN "I", #2, "DATAFILE"
  '
  '
>200 LINE INPUT #2, X$
  '
  '
>300 END
```

2)

```
>10 OPEN "O", #1, "SEQ.DAT"
>20 PRINT #1, "BASIC, 8086"
>30 CLOSE
>40 '
>50 OPEN "I", #1, "SEQ.DAT"
>60 LINE INPT #1, A$
>70 PRINT A$
>80 CLOSE

>RUN

BASIC, 8086
```

LINE INPUT%

Purpose

The LINE INPUT% has the same format as the LINE INPUT and reads the character data from the RS-232C port into the variable.

Form

LINE INPUT% <port number>, <string variable>

Description

(1) <port number> takes only 1 corresponding to the line connector socket on the rear panel.

(2) The data is read out of the serial port into <string variable> regarding the whole string as one data item, even if a comma (,) is included. The read-out operation terminates when <CR> appears in the string.

Example

```
400 LINE INPUT%1, A$
```

LIST/LLIST

Purpose

The LIST statement outputs the program, which has been currently stored in memory, on the screen.

Form

LIST [[<Line Number 1>] - [<Line Number 2>]]
LLIST [[<Line Number 1>] - [<Line Number 2>]] [,[<integer constant>]]

Description

(1) The program which has been stored in memory is output on the screen by the LIST statement or to the printer by the LLIST statement.

(2) The specification of <line number> is interpreted as follows:

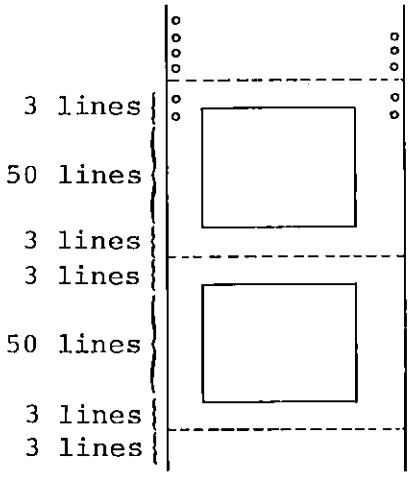
1.	Not specified	Outputs the whole list.
2.	<Line Number 1> - <Line Number 2>	Lists from <Line Number 1> up to <Line Number 2>
3.	<Line number> -	Lists from the line specified by line number up to the end of the program.
4.	-<Line number>	Lists from the beginning of the program up to the line with the specified line number.
5.	<Line number>	Lists the data of the specified line only. A period (.) indidates the final line of the program.
6.	LIST	Lists the data of the final line of the program.

(3) <Line number> is written with an integer constant only.

(4) When <integer constant> is written in the LLIST statement, one pageconsists of the number (integer constant) of lines. Six blank lines are aligned between pages.

Example

>LLIST, 50 -->



(5) The integer constant to specify the number of lines for one page should be in the range between 0 and 255. when it is 0, the lines are output with no page alignment. (It produces the same effect as the statement with no specification.)

(6) When <integer constant> is omitted, (LLIST,) 60 lines with following 6 blank lines are assigned for one page. (The 11" sized stock form is assumed.) If the <integer constant> option is not made at all, the lines are output with no page alignment, and no blanks are sent between pages.

(7) Listing stops by pressing the space bar and resumes by pressing any key.

(8) When a program is printed out, all special characters as shown in the character code table (Column 8, 9, E, F) are printed as a question mark (?).

Examples

>LIST	Lists the whole program on the screen.
>LIST 100-500	Lists the data from Line 100 to 500 on the screen.
>LIST - 250	Lists the programs from the beginning to Line 250.
>LLIST 100- 1000,	Prints out the data from Line 100 to 1000 with 60 lines per page.
>LLIST 1000- ,50	Prints out the data from Line 1000 and further with 50 lines per page.
>LLIST	Prints out the whole program with page alignment but with no blank lines.
>LIST.	Outputs the final line only.

LOAD

Purpose

LOAD reads the program out of the disk into memory.

Form

LOAD <file name> [,R]

Description

(1) <file name> specifies the disk file in which the program has been saved.

(2) When the LOAD statement is executed, the system closes all the files currently opened and destroys the program presently in the memory, and loads the program of <file name>. After execution the control returns to the BASIC Mode.

(3) When ",R" is included, the system keeps the files open, reads the prograı from the disk, and immediately executes the entered program.

(4) When there is no corresponding file on the disk, which is specified by <file name>, the current programs remain intact in memory.

Examples

```
>LOAD "PRGM-A"
>LOAD "PRGM-A",R
```

LOCATE

Purpose

The LOCATE sets the cursor to any desired position on the screen.

Form

LOCATE <Expression 1>, <Expression 2> [,<Cursor blink switch>]

Description

(1) <expression 1> and <expression 2> constitutes the coordinate to show the Cursor location. the coordinate has 24 line pointers vertically and 80 character pointers horizontally. The location of a Cursor can be specified within this range only.

(2) <expression 1> indicates the horizontal position (line pointer) in the range between 1 and 80. (1 means the left side and 80 the right side.)

(3) <expression 2> indicates the vertical position (line pointer) in the range between 1 and 24. (1 means the tope and 24 the bottom.)

(4) When the evaluation value of <expression> includes a decimal part, it is automatically rounded (counting fractions 5 and over as a unit and disregarding the rest). If the value falls out of the range, and error occurs.

Example

```
10 X=30
20 Y=10
30 A$="HERE"
40 LOCATE X, Y
50 PRINT A$
60 END
```

LPRINT/LPRINT USING

Purpose

These two kinds of statement output the data to the printer. They work similar to the PRINT and PRINT USING statements.

Form

LPRINT [<expression> {,|; <expression>} [,|;]]

LPRINT ISOMG <character string> ; <expression> {,<expression>} [,|;]

Description

(1) For an explanation of these statements, refer to the PRINT and PRINT USING statement respectively.

LSET/RSET

Purpose

When a random file is used, the LSET/RSET statement moves the data into the buffer area before the data is written to the file.

Form

```
LSET <string variable> = <string expression>
RSET <string variable> = <string expression>
```

Description

(1) <string variable> specifies the name of the buffer area and is defined by the FIELD statement.

(2) <string expression> is the data to be written to the file and replaces <string variable> of the buffer area starting from the left end by the LSET statement or from the right end by the RSET statement. When the number of characters of <string expression> exceeds the value defined by the FIELD statement, the remainder is disregarded. When it is less than the defined value, the spaces are filled up.

(3) When numeric data is used, the numeric-character conversion function (MKI$, MKS$, or MKD$) is specified before preceding to the LSET RSET statement to convert the numeric data to string data.

Example

```
  '
>100 OPEN "R",#1, "CUSTFILE", 30
>110 FIELD #1, 10 AS A$, 20 AS B$
  '
  '
>200 X$="CUSTOMER-1"
>210 Y$="12345"
  '
  '
>300 LSET A$=X$
>310 RSET B$=Y$
  '
  '
>350 PUT #1, N
```

MERGE

Purpose

This statement works for merging the current program with a particular program in the disk.

Form

MERGE <file name>

Description

(1) <File name> indicates the program file saved on the disk, which will be merged with the current program.

(2) When the program with the specified <file name> has the same line number as the current program, the former will take over the latter in memory.

(3) The control returns to the BASIC Mode when the MERGE statement has been executed.

Example

```
MERGE "PRGM-A"
```

MID$

Purpose

The MID$ statement modifies a part of the string data with another character string.

Form

MID$ (<string variable>, <numeric Expression 1> [,<numeric Expression 2>]) = <string expression>

Description

(1) MID starts with the first <Numeric Expression 1> character and replaces it with the <Numeric Expression 2> characters from <string expression>.

(2) <Numeric Expression 1> and later characters of <string variable> are to be replaced with the first to <Numeric Expression 2> characters of <string expression>.

(3) When <Numeric Expression 2> is omitted, all the characters of <string expression> are used for replacement.

(4) The length (total number of characters) of <string variable> does not change.

```
> 100 GOSUB 200
> 110 MID$ (A$, 5, 3) =B$
> 120 PRINT A$
> 130 GOSUB 200
> 140 MID$ (A$, 8)=B$
> 150 PRINT A$
> 160 END
> 200 A$="ABCDEFGHIJ"
> 210 B$="12345"
> 220 RETURN

>RUN

BCD123HIJ

ABCDEFG123
```

NAME

Purpose

A file name can be changed with NAME.

Form

NAME <old file name> AS <new file name>

Description

(1) <Old file name> is replace by <new file name>.

(2) <Old file name> must exist in the disk but <new file name> may not.

Example

```
NAME "FILE-A" AS "DATA-1"
```

NEW

Purpose

The NEW deletes the current program in memory.

Form

>NEW

Description

(1) The NEW statement should always be executed when a new program is entered through the keyboard. Otherwise, parts of the old program may be left in the new program.

(2) The control returns to the BASIC Mode after the execution of the NEW statement.

Example

```
>NEW
```

ON ERROR GOTO/RESUME

Purpose

The ON ERROR GOTO statement sends program control to the error handling subroutine when an error occurs during execution of the program.

Form

```
ON ERROR GOTO <line number>
.....
RESUME [NEXT | <line number>]
```

(1) <line number> indicates the destination after an error occurs. the lines after this line can be regarded as the error handling routine.

(2) The ON ERROR GOTO statement can be written anywhere in the program as many times as the user likes. When an error occurs, the control jumps to the <line number> of ON ERROR GOTO which has been previously written.

(3) When and error occurs in the program which does not have the ON ERROR GOTO statement, the system displays the error message and returns the control to the BASIC Mode. but by writing the ON ERROR GOTO statement, the user can set the control for erroneous program processing.
This instruction does not work for an error during the execution of the error handling routine. In that case, the system displays the error message and stops the program handling.

(4) When an ON ERROR GOTO 0 is entered, the error handling routine becomes invalid. No more processing of the error handling routine will happen.

Examples

1)

```
   '
   '
> 100 ON ERROR GOTO 310
   '
   '
>150 A=B/C
   '
   '
>300 END
>310 IF ERR=11 THEN C=1: RESUME
```

2)

```
>100 ON ERROR GOTO 160          ....The program for overflow
                                    processing
>110 input a#
>120 A%=A#
>130 PRINT A%
>140 END
>150 '
>160 IF ERR=6 THEN A%=32767
>170 RESUME NEXT

>RUN

? 1234567890

  32767
```

ON GOSUB/ON GOTO

Purpose

This is a conditional diverging instruction to specify the destination with a particular value in the program. After execution control returns to the statement after the ON GOSUB statement by means of the RETURN instruction at the end of the subroutine.

Form

ON <numeric expression> GOTO <line number> {,<line number>}
ON <numeric expression> GOTO <line number> {,<line number>}

Description

(1) The evaluation value of <numeric expression> specifies the line to which the control jumps. <Line number> must be written with an integer constant.

(2) The value of <numeric expression> should be in the range between 1 and the number of <line number>'s. when it includes a fraction, it is automatically rounded (counting the fractions 5 and over and disregarding the rest). When it is negative or exceeds 255, an error occurs.

(3) When the value of <numeric expression> is 0 or exceeds the number of <line number>, this statement will not be executed and the control moves to the next statement.

(4) The ON GOSUB statement <line number> calls the starting line of the respective subroutine.

Examples

1)

```
10 A = 3
20 B = 2
30 ON A+B GOTO 110, 120, 130, 140, 150, 160

(A + B = 5, control goes to Line 150 which is
 the 5th number.)
  '
  '
100 END
  '
  '
```

2)

```
> 100 INPUT "1-3?", A
> 110 ON A GOSUB 140,170,200
> 120 GOTO 100
> 130 '
> 140 PRINT "A=1"
> 150 RETURN
> 160 '
> 170 PRINT "A=2"
> 180 RETURN
> 190 '
> 200 PRINT "A=3"
> 210 RETURN

>RUN

 1-3 ? 2

A=2

 1-3 ? 100

A=1

 1-3 ?

Break in 100
```

OPEN

Purpose

To use the disk files, the OPEN opens the data files in the disk. Reference to Section J is also advised.

Form

OPEN <mode>, [#] <file number>, <file name> [,<record length>

Description

(1) One of the following four characters is assigned to <mode>:

1.	"O"	(Output)	Output mode of the sequential files.
2.	"I"	(Input)	Input mode of the sequential files.
3.	"A"	(Append)	Append mode of the sequential files.
4.	"R"	(Random)	Input/output mode of the random files.
5.	"IS"		Input/output mode of the index sequential file.

(2) <File number> should be in the range between 1 and the usable file number specified at the time of initialization of the BASIC program. <Numeric expression> can be written for <file number>.

(3) <File name> specifies the file to be used.

(4) <Record length> is the number of bytes for one record of data. Its specification is required only in the "R" mode. The record length can be specified freely in the range between 1 and 128 bytes. When omitted, the default value "128" is assumed.

(5) Sequential files should be opened after write-in or read-out operation is selected. For the random files, both modes become valid by writing the OPEN statement once.

(6) "A" mode means the continuous output mode of the sequential file. When specified, new data can be written continuously following the existing data on the file.

(7) When the file is opened, the file name becomes coupled with its number and can be referred by simply writing the file number in other statements. this relationship is maintained until the file is closed. when closed, the user can specify the file number to be related with another file name and open the new file.

(8) The file which is used for input in "I" or "R" mode can be opened again with the different line numbers. A file can be opened for "I" mode or "R" mode for input on more than one file number at a time. For output, however, a file may be opened on only one file number at a time. The file used in the output mode can be opened with the particular line number only.

Example

```
   :
   :
>100 OPEN "R", #1, "FILE-1.DAT", 80
>110 FIELD #1, 20 AS A$, 10 AS B$, 30 AS C$
   :
   :
>200 END
```

OPTION BASE

Purpose

This determines the minimum value of the subscript to be attached to an array.

Form

OPTION BASE <n>

Description

(1) <n">is either 1 or 0.

(2) When the array declaration is made by DIM statement, the elements from 0 to the declared number are reserved normally. When DIM A(20) is declared, up to 21 elements (A(0) to A(20) are reserved in memory. The OPTION BASE is used in case the user likes to reduce the number of elements (20 elements between A(1) and A(20)).

(3) Only one OPTION BASE statement is written in a program. It is required to be executed before any of the DIM declarations. (Arrays with Base 0 and 1 must not be mixed in a program.)

(4) Also, when the programs are chained by the CHAIN statement, the arrays of the chained programs should have the same base of either 0 or 1.

Example

```
>10 OPTION BASE 1
>20 DIM A(5)
>30 PRINT A(0)

>RUN

Subscript out of range in 30
```

OUT

Purpose

This sends the data of 1 byte to the output port of the processor. If nothing is connected to the output port, operation of the computer is unpredictable.

Form

OUT <Numeric Expression 1>, <Numeric Expression 2>

Description

(1) <Numeric Expression 1> produces the port number with an integer between 0 and 255.

(2) <Numeric Expression 2> indicates the data to be sent, of which value will be an integer from 0 to 255.

POKE

Purpose

The POKE writes the 1-byte data into the specified address of memory.

Form

POKE <Numeric Expression1> , <Numeric Expression 2>

Description

(1) <Numeric Expression 1> specifies the memory address to which the data is written. the evaluation value should be in the range between 0 and 65535.

(2) <Numeric Expression 2> specifies the data to be written and should result in the value between 0 and 255.

(3) When both expressions result in real numbers, they are automatically rounded (counting the fraction 5 and over as a unit and disregarding the rest). If it falls out of the range, an error occurs.

Examples

1)

```
>10 POKE &HA000, &HFF (FF entered in Address
                       &HA000.)
  '
  '
>120 FOR I = 0 TO &HFF
>130 POKE &H8000 + I, &H20
>140 NEXT I              (A space code is entered
  '                       from 8000 through to
  '                       80FF.)
>200 END
```

2)

```
>10 A%=32767                .......This process modifies the
                                    variable contents by directly
                                    writing into the memory.
>20 PRINT A%
>30 ADD=VARPTR(A%)
>40 POKE ADD, 0
>50 POKE ADD+1, 0
>60 PRINT A%

>RUN

32767

0
```

PRINT

Purpose

The evaluation value of <expression> is displayed on the screen using numerics or characters.

Form

PRINT [<expression> {,1; <expression>} [,1;]]

Description

(1) When the delimiter is a comma: A line (consisting of 80 characters) on the screen is divided into five zones having 16 characters each, and the evaluation value of <expression> after the comma is output in the following zone.

(2) When the delimiter is a semicolon: The evaluation value of expression after the semicolon is output immediately following the value of the previous expression.

(3) When the comma or semicolon at the end of the line is omitted:

1. <CR> is sent at the end of the PRINT statement on the screen.
2. The Cursor is returned and moved to the left of the next line, and
3. The new line is entered deleting the old line.

When the comma or semicolon exists at the end of the line:

1. <CR> is not sent to the screen, and
2. In the case of a comma, the output of the display continues from the next zone (tab) or, in the case of a semicolon, the display continues immediately after the previous part.

Example

```
>PRINT 10, 100, 0.1, -20
 10      100        .1      -20


>PRINT 10;100;0.1;-20
 10△△100△△0.1△-20


>A=10: B=20: PRINT "A=";A, "B=";B
 A=△10      B=△20
```

PRINT USING

Purpose

Display data on the screen in an edited format.

Form

PRINT USING <string expression>;<expression>{,1;<expression>} [,1;]

Description

(1) The evaluation value of <expression> is output on the screen in the format specified with <string expression>. The effect of a comma <,) or semicolon (;) is similar to the PRINT statement.
(2) The formats which <string expression> specifies are as follows. <LF> cannot be included in the value of <string expression>.

A. When <expression> is string (output of a character field):

1. "!" Outputs the first character of the value of <expression>.

```
>10 A$ = "JOHN"
>20 B$ = "PAUL"
>30 C$ = "JONES"
>40 PRINT USING "!"; A$;".";B$;".";C$;"."
>RUN
J.P.J.
```

2. "& (n) &" Outputs up to the (n+2)th characters of the value. When the digits of characters of <expression> is smaller than (n+2), blanks are assigned to the remainder.

```
 10 A$ = "JOHNSON"
 20 B$ = "PAULIS"
 30 C$ = "JONEVILLE"
 40 PRINT USING "&  &";A$;" ";B$;" ";C$
>RUN
JOHN△△△△ PAUL △△△△ JONE
```

B. When <expression> is a numeric expression (output of a numeric field):

1. # Specifies the pointer position for the output of characters. Validated

characters are output starting from the right end of the display line, and when more spaces exist in the leftward part, blanks should be keyed in. The numeric characters can be written into the specified

2. . (period) A decimal point can be included in the numerics.

```
>10 A = 0.536
>20 B = 3.1416
>30 C = 23.1
>40 PRINT USING "##.##"; A, B, C
>RUN
△0.54       △3.14       23.10
```

3. +, - The signs ("+" and "_") can be written as a prefix or suffix to the numerics.

```
>10 A = 0.536
>20 B = 3.1416
>30 C = 23.1
>40 PRINT USING "+###.#"; A, B, C
>RUN
△△+0.5    △△+3.1     △+23.1
```

When the specification of a sign to be included in both the prefix and suffix is made, the sign of the number takes precedence over the prefix.

```
>PRINT USING "+###.#-"; -12.5
△-12.5-
```

4.** "*" is assigned to blank character to indicate the number of digits to be reserved in the field.

```
>10 FOR I = 0 TO 150 STEP 50
>20 PRINT USING "**##"; I: NEXT
>RUN
***0
**50
*100
*150
```

5. \\ A "\" can be written in the prefix of

the numeric. Slashes "\" can be used to specify digits and are reserved in the field.

```
>10 PRINT USING "\\###"; 50, 100, 500
>RUN
,Δ,,Δ,\50     ,Δ,\100       -\500
```

6. **\ Both "**" and "\\" can be written. The numeric of a certain number of digits and **\ are saved in the field.

```
>10 PRINT USING "**\###"; 50, 1000, -12.6
>RUN
***\50      *\1000      **-\13
```

7. , (comma) A comma can be written. If it preceeds the decimal point it is output in every fourth digit. If it is written after the decimal point, it is regarded as a character. The number of digits are reserved in the field including the commas.

```
>10 PRINT USING "#########,.##";1234567.8
>RUN
Δ1,234,567.80
```

8. Specifies the format for exponential representation. The numeric with a valid number of digits is output starting from the left end, and its exponential part is written accordingly. As long as the numeric does not include a "+" in the prefix, or "+" or "-" in the suffix, either a "-" sign or a space will be written at the left side of the fraction point.

```
>10 PRINT USING "#.###^^^^"; 12345, -12345
>RUN
Δ.1234E+05        -.123E+05
```

If Methods 7 and 8 are used concurrently, the comma becomes invalid.

Or, when the method is taken together with "\", and error occurs.

When characters other than those above are written in the string of the format specification, they are output as they are written.

```
>10 PRINT USING "APPLES COST ##.## $"; 3.56
>RUN
APPLES COST   3.56 $
```

PRINT#/PRINT# USING

Purpose

These two kinds of statements instruct the same type of operation as the PRINT/PRINT USING statement but the data is output to disk files.

Form

```
PRINT#<File number>,CUSING<character string>;]
<expression>{,|;<expression>}
```

Description

(1) <File number> specifies the number under which the file was opened for output.

(2) The file with the specified file number should have been opened in the "0" Mode (Sequential File Output Mode).

Example

```
  '
  '
>100 OPEN "0", #3, "FILE"
  '
  '
>200 PRINT #3, A, B, C
>210 CLOSE #3
>220 END
```

PRINT%/PRINT% USING

Purpose

These two kinds of statments instruct the same type of operation as the PRINT/PRINT USING statement but the device to which the data is output will be the serial port of RS-232C.

Form

```
PRINT%<port number>[USING <character string> ;]
<expression>{,|;<expression>}
```

Example

```
>10 A = 0.536
>20 B = 3.1416
>30 C = 23.1
>40 PRINT%1, USING "##.##"; A, B, C
>50 END
```

PUT

Purpose

The PUT statement instructs the write-in operation into the random file. with this specification the data which has been saved in a buffer is entered into the random file.

Form

```
PUT [#]<file number>[,<record number>]
```

Description

(1) <file number> specifies the number under which the file was opened for output. The file with the specified file number should have been opened in the Random Mode.

(2) <record number> indicates which record saves the data to be entered. When this specification is omitted, 1 is added to the record number of the PUT or GET statement which has been executed immediately before. The specified record number must fall in the range between 1 and 32767.

(3) The form for index sequential files is:

```
PUT [#] <file number>,<key>
```

A) <file number> is used to open the file to be written. This can be specified using a numeric expression. The file must be opened with "IS" mode

B) <key> is the key of the data to be written. This can be specified using a character expression.

C) If the key already exists in the file, the data is replaced by the latest contents.

D) If the number of records exceeds the number defined by the CREATE statement, the data cannot be written.

Example

```
  '
  '
>100 OPEN "R", #L, "CUST.FIL", 30
>110 FIELD#1, 10 AS A$, 20 AS B$
  '
  '
>150 X$ = "CUSTOMER-1"
>160 Y$ = "10001"
  '
  '
>200 LSET A$ = X$
>210 RSET B$ = Y$
  '
  '
>300 PUT #1, REC%
  '
  '
>400 END
```

RANDOMIZE

Purpose

The RANDOMIZE modifies the sequence of pseudorandom numbers that are created by RND function, and produces another sequence.

Form

RANDOMIZE <expression>

Description

(1) The evaluation of <expression> must result in an integer from -32768 to 32767.

(2) A different random number array is created after the execution of the RANDOMIZE statement through the evaluation of <expression>.

(3) Without the RANDOMIZE statement, the same sequence of random numbers (RND) will be produced each time the program is run from system. Since the random numbers are infinite, the same string will not occur while the program is running.

READ/DATA/RESTORE

Purpose

The READ and DATA statements are used together; the constant specified in the DATA statement is assigned to the vairable specified in the READ statement. The RESTORE statement moves the pointer to a specified line of DATA statements.

Form

```
READ <variable>{,<variable>}
DATA <constant>{,<constant>}
```

Description

(1) <Variable> is a string or numeric variable and corresponds to each <constant> in the DATA statement. More than one READ statement can be written for one DATA statement or vice versa. the variables of the READ statement are read in order starting from the top to the bottom, in which each of the variables corresponds to each of the constants. When the number of variables in the READ statement exceeds that of the constants in the DATA statement, an error occurs. When the number of constants exceeds that of the variables, the rest of data in the DATA statement is disregarded.

(2) The DATA statements can be written anywhere in the program as many times as the user likes. the READ statement reads constants of the DATA statements in the sequence of their line numbers.

(3) When the variable in the READ statement is of numeric data and the corresponding constant in the DATA statement is of string data, an error occurs.

(4) When both the variable and constant are of numeric data but their types are different, the constant is converted to the variables type and read in. When the conversion is impossible, an error occurs.

(5) If the user wants to write commas (,), colons (:), or spaces within the character constant, that constant should be enclosed in quotation marks (" "). The user need not use " " for the character constant in which no delimiter exists.

(6) Constants in the DATA statement are used sequentially as they appear, however, the RESTORE statement can change the sequence for a read-in operation.

```
RESTORE [<line number>]
```

(7) When the RESTORE statement is executed, the DATA statement which corresponds to the READ statement to be executed next, has its line number specified in the RESTORE statement. When the line number is omitted, the smallest line number is assumed for the DATA statements. If the specified line number is not found among the DATA statements, the read-in pointer is set to the first appearing DATA statement after the line specified by <line number>.

Examples

1)

```
>10 DIM A(10)
>20 FOR I=1 TO 10
>30 READ A(I)
>40 PRINT A(I)
>50 NEXT I
>60 DATA 1,2,5,10,3,7,15,8,3,4
>70 END
>RUN
1
2
5
'
'
4
```

2)

```
>10 READ A,B,C,D,E,F
>20 RESTORE 50
>30 READ G,H,I
>40 DATA 10,10
>50 DATA 20,20
>60 DATA 30,30
>70 PRINT A,B,C,D,E,F,G,H,I
>80 END
>RUN
10  10  20  20  30  30  20  20  30
```

REM

Purpose

The REM statement declares a comment to be entered in the BASIC source program. There is no effect on the execution of the program.

Form

```
REM <comment>
      or
'   <comment>
```

Description

(1) Any of the usable characters can be written in <comment>.

(2) Any character can be written in <comment>, if it is admitted in the list of Section D-1. The system executes the program disregarding comments.

(3) When the REM declaration is seen in a Multiple Statement Line, the whole part beginning with REM and ending with <CR> is regarded as a comment.

(4) An apostrophe can be used instead of REM.

Example

```
10 REM *** RRGM-2 ***
20 '   1983.1.1
30 '
40 '   BY J. Thomas
50 '
60 PRINT "THE REM STATEMENT HAS NO EFFECT ON
EXECUTION."
70 END
```

RENUM

Purpose

All the line numbers in memory can be modified at the same time by use of the RENUM command.

Form

RENUM [[<new line number>] [,[<old line number>] [,<increment>]]]

Description

(1) <New line number> is the first number to be nenewed. <Old line number> indicates the starting line from which the renumbering operation should be performed.

(2) <Increment> shows the interval for renumbering.

(3) When <new line number> and <increment> are omitted, the default value 10 is assumed for each of them. When <old line number> is omitted, the renumbering starts from the first line stored in memory.

(4) An error occurs when:

1. No program to be renumberd exists in memory
2. The new line numbers exceed 65535
3. The sequence of program flow is changed by the RENUM statement
4. 0 is assigned to <increment>

(5) The number of characters in one line may exceed 255 by using the RENUM statement. The program will work and can be listed out. However, when the line is modified in the Screen Edit Mode, an error occurs if the line length is greater than 255 characters. The user is required to modify the line to be within 255 characters.

Examples

Old Line Number	RENUM	RENUM 100	RENUM 70,52,20
10	10	100	10
30	20	110	30
50	30	120	50
52	40	130	70
56	50	140	90
70	60	150	110
90	70	160	130

RESET

Purpose

The RESET statement instructs certain processing to be performed before removing the disk out of the drive unit.

Form

RESET

Description

(1) When the RESET statement is executed, the system closes all the open files, writes the index data in the index track, and resets the unit write protect.

(2) The RESET statement should be specified when the user removes the disk before the program ends normally. If not done, the right data may not be read out of the disk again.

RESUME

Purpose

Control returns from the error handling subroutine to the main program by use of the RESUME statement.

Form

RESUME [NEXT <line number>]

Description

(1) The RESUME statement is used together with the ON ERROR GOTO statement. (When either of them is omitted, an error occurs.) If an error occurs, the control jumps to the error handling subroutine by the ON ERROR GOTO statement. When the handling is terminated, it returns to the line of the main program which is specified in the RESUME statement.

(2) When the line number is not specified, the control returns to the statement where the error has occurred. When NEXT is written, it returns to the statement after the erroneous one, and when <line number> is specified, to the line having the specified number.

Example

```
>10 A = 15: B = 35
>20 ON ERROR GO TO 300
>30 A$ = 10                (An error occurs)
  '
  '
>140 PRINT A + B
>150 END
  '
  '
>300 IF ERR = &H0D THEN A = 1
>310 RESUME NEXT
```

RUN

Purpose

The RUN statement performs the execution of a program.

Form

(1) RUN [<line number>]

(2) RUN <file name> [,R]

Description

(1) Format (1) performs the execution of a program which has been loaded in memory. When <line number> is specified, the execution starts with the specified line; when omitted, it starts with the first line of the program.

(2) The format (2) is to load the program with the specified file name from the disk to memory for execution. When option "R" is specified, the system loads and executes the data program keeping the files open.

Example

```
>10 DIM A$(15)
  '
  '
>90 FOR I = 1 TO 15
>100 PRINT A$(I)
>110 NEXT I
>120 DIM B$(15)
>130 FOR J = 1 TO 15
>140 PRINT B$(J)
>150 NEXT J
>160 END
>RUN 120  (Execution will begin with LINE 120)
```

SAVE

Purpose

The SAVE statement stores the current program in memory to the particular disk file.

Form

SAVE <file name> [,A|P|Q]

Description

(1) A <file name> is assigned to the program. It can be called by the LOAD statement and loaded again in memory.

(2) An "A" indicates the ASCII coding mode, and the program will be written in the disk as it is. When P is specified, the program is converted to the intermediate code for write operation to the disk file.

(3) If neither specification is made in the statement, P is assumed.

(4) By using Q, the program cannot be listed nor can it be copied.

Example

```
>SAVE "FILE-1"
>SAVE "FILE-2", A
>SAVE "FILE-3", Q
```

SET

Purpose

The SET statement defines the attributes for disk files.

Form

SET #<file number> | <File name> | <unit>, <attribute symbol>

Description

(1) The file attributes are divided into the following two types:

W Write Protect (Read Only)
Prohibits the write operation. Nothing can be written in the file for which "W" attribute is set.

S System (Invisible)
Nothing can be read out of the file by the FILES/LFILES statement, for which "S" attribute is set.

(2) <File number> specifies the open file, and <file name> is the name of that particular file. <Unit> shows the name of a drive unit; i.e., either A: or B: (C: or D: can be also used when optional units are installed). By specifying these three items, the attribute can be set.

(3) The Attribute can be reset by the SET statement with "N" specification.

NOTE A file can be destroyed if a reset of attributes is attempted while the file is open. Files should always be closed before reset.

Examples

1)
```
'
'
250 OPEN "R", 1, "FILE", 60
260 FIELD #1, 20 AS A$, 10 AS B$, 30 AS C$
'
'
500 CLOSE
510 SET "FILE-1", W
'
'
600 END
```

2)
```
- SET "FILEA.DAT",W
- SET "FILEA.DAT",N      ......RELEASING THE ATTRIBUTE
- SET A:,W
- SET #3,S
- SET B:,N               ......RELEASING THE ATTRIBUTE
```

SETKEY

Purpose

SETKEY sets the record pointer of the index sequential file immediately before the specified key.

Form

SETKEY [#] <filename> [,<key>]

Description

(1) <filename> is used to open the file to be read. Use a numeric expression. The file must be opened with "IS" mode.

(2) <key> is the key of the data to be set. Use a character expression. An error occurs if the key of the data which is not PUT is specified. If the key is omitted, the pointer becomes 0 and reads the first record when the GET without key is executed.

STOP

Purpose

With the STOP command, the system stops the program execution temporarily and the control returns to the BASIC Mode.

Form

STOP

Description

(1) The STOP command can be written in any part of a program, and when it is executed, the system displays the line number where the processing is halted:

Break in Line XXXX

The control then returns to the BASIC Mode.

(2) The program stopped by the stop statement can be restarted from the next line by use of the CONT statement.

(3) The STOP command does not close the files.

SWAP

Purpose

This statement swaps the contents of two different variables.

Form

SWAP <variable 1>, <variable 2>

Description

(1) The contents of <variable 1> and that of <variable 2> are interchanged.

(2) The type of both variables should agree with each other.

Example

```
10 INPUT "ENTER VALUES FOR A AND B."; A, B
20 IF A > B THEN SWAP A, B: GOSUB 60
30 C = B * (B + A)
40 PRINT C; " IS THE VALUE WHEN A < B."
50 END
60 REM GOSUB ROUTINE
70 C = A * (A + B)
80 PRINT C; " IS THE VALUE WHEN A > B."
90 RETURN
```

SYSTEM

Purpose

The SYSTEM statement returns the control to the Operating System Mode.

Form

>SYSTEM

Description

(1) When the SYSTEM instruction is executed, the control returns to the Operating System Mode closing all the open files, if any.

TINPUT

Purpose

This is an input instruction through the keyboard. The INPUT statement displays the input data by shifting the Cursor right one column, the TINPUT statement on the other hand shifts the input data leftward and keeps the Cursor where it is. (The display of the line starts from the right end.) The TINPUT statement is used expecially for the input of numeric data.

Form

```
TINPUT [(<number of digits>)] [;]<variable>
```

Description

(1) When displaying the input data on the screen, the TINPUT statement specifies the cursor position to be at the end of the line and starts the line display from the right end.

(2) The input data is shipted left one column every time a new character is keyed in. The Cursor does not move.

(3) When the first keyed in character reaches the left end, only <CR> can be input, and other characters, even though keyed in, are disregarded.

(4) When the variable does not agree with the keyed-in constant, the system does not stop but waits for valid data to be entered, clearing the data so far entered.

(5) <number of digits> is thenumber of digits of the keying data used to check and can be written using a numeric expression. An integer 1 through 255 can be specified. The fraction part is automatically rounded. The number is indicated as an error if it exceeds the limit. When <number of digits> is specified, up to the number of digits can be entered, although only the RETURN key can be entered after that. All other input is ignored.

(6) When a semicolon (;) is placed between the TINPUT and <variable>, <CR> to show the termination of a line is not sent to the screen and the Cursor also does not move. If the semicolon is omitted, the Cursor is returned and fed to the next line.

(7) To use the TINPUT statement effectively, the cursor should be set at the proper position in advance assuming the number of characters of data to be input. (For this purpose, use the LOCATE or PRINT statement.

The TINPUT statement works similar to the INPUT statement.
in other cases.

Examples

1)

```
10 LOCATE 20, 5
20 TINPUT A$
30 PRINT A$
40 END
```

2)

```
- TINPUT;X
```

TRON/TROFF

Purpose

These statements instruct the set and reset of the Trace Mode for program debugging.

Form

```
TRON [P]
TROFF
```

Description

(1) When the TRON instruction is used (to enter the TRACE Mode), all the line numbers of the program that have been executed so far are displayed on the screen from the very beginning of the specification. When the TROFF instruction is executed, the control returns to the normal execution mode.

(2) Even in the Direct Mode, the TRON/TROFF instruction is valid for the further execution of the program.

(3) When TRON P is written, the system outputs the traced line numbers on the printer.

(4) The Trace Mode is released when the NEW instruction is conducted.

Example

```
 TRON
 10 A = 45
 20 B = 56
 30 C = A + B
 40 PRINT A, B, C
 50 END
>RUN
<10><20><30><40> 45       56       101
<50>

 TROFF
>RUN
 45             56             101
```

WAIT

Purpose

The WAIT command controls the program execution from the input ports of CPU (Central Processing Unit).

Form

WAIT <port number>,<Expression 1>[,<Expression 2>]

Description

(1) When the WAIT instruction is conducted, the execution of the program is held until the particular bit pattern of the CPU's input ports, whose number is specified with <port number>, is seen.

(2) An exclusive logical sum of the data from the input port and the value of <Expression 2> is made first, then a logical product of the sum result and the value of <Expression 1> is made.

(3) If the logical product becomes 0, the data is read out of the input port again. These procedures are repeated. When the logical product is not 0, the control of the program moves to the next step. In case <Expression 2> is omitted, the default value "0" is assumed.

(4) The evaluation results of <Expression 1> and <Expression 2> should be integers between 0 and 255.

WHILE/WEND

Purpose

These statements instruct the repetition of statements to be performed consecutively until a specified condition is not met.

Form

```
WHILE <expression>
.............
WEND
```

Description

(1) While the conditions given by <expression> are met, the statements up to WEND are repeatedly executed. The difference between the WHILE/WEND statement and the FOR/NEXT statement is that the judgement of conditions is done before the statements are looped.

(2) If the evaluation of <expression> results in 1 (the expression is valid), the group of statements from the next line to WEND are executed. When the control reaches the WEND statement, it returns to the WHILE statement for re-evaluation of <expression>. In case <expression> is valid, the loop is executed again. In case it is evaluated as 0 (the expression is not valid), the control jumps to the statement after to WEND.

(3) The WHILE and WEND statements can be nested. The WEND statement can be nested in the preceding WHILE statement according to the line sequence.

Examples

1)

```
'
'
100 WHILE A > 0
110 X = X + Y
120 A = A - 1
130 WEND
```

2)

```
10 WHILE A<9999
20 INPUT A
30 PRINT A
40 WEND

>RUN

?505

 505

?1000

 1000

?9999

 9999
```

WIDTH

Purpose

The WIDTH statement determines the output length (the number of characters in one line) of the printer.

Form

WIDTH [<output width>]

Description

(1) When the WIDTH is executed, the system line feeds every time the specified number of characters fill up the line space. The number of characters is provided by the evaluation value of <output width>.

(2) The evaluation value of <output width> should be an integer from 30 to 255. When it is not an integer, the fraction part will be rounded (counting the fractions 5 and over as a unit and disregarding the rest). If it falls out of the range, an error occurs.

(3) When the line length is not specified with the WIDTH statement, one line for printing is set with up to 80 characters in SANYO-BASIC.

(4) If the evaluation results in the value 255, the length is interpreted as unlimited, and the system does not send <CR> code to the printer. The printer head position, in this case, goes over 255 and 0 is returned with the LPOS function.

Examples

```
10 WIDTH 30
20 '
30 FOR I=1 TO 100
40 LPRINT "A";
50 NEXT

>RUN

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAA
```

WRITE

Purpose

Similar to the PRINT statement, the WRITE command instructs the output of data on the screen. However, when the WRITE command is specified, commas are automatically placed between the output items. String data is enclosed automatically in quotation marks (" ").

Form

```
WRITE [ expression>{,<expression>}]
```

Description

(1) The value of <expression> appears on the screen. If there are a number of <expression(s)>, each <expression> is delimited with a comma but no space follows. When <expression> is numerical, zero suppression occurs.

(2) In case of a string <expression>, the character string output on the screen is automatically enclosed in quotation marks (" ").

(3) When <expression> is omitted, the line is simply fed. (A blank line is output on the screen.)

Example

```
 10 A$ = "P.107"
 20  B$ = "SANYO"
 30 C$ = "USER'S GUIDE"
 40 WRITE A$, B$, C$
>RUN
"P.107","SANYO","USER'S GUIDE"
```

WRITE#

Purpose

The WRITE# command instructs the output of data to the disk (sequential) files, and the data is written in the disk files in the format similar to the WRITE command. The commas as well as quotation marks are added automatically; therefore, the delimiters can be easily written in the disk files.

Form

WRITE#<file number>,<expression>{,<expression>}

Description

(1) <File number> indicates the file which is open for write-in operation. The file to be used should have been opened in either "O" or "A" mode.

(2) See other sections for the particular usages of PRINT#, WRITE#, INPUT#, and LINE INPUT# instructions.

Example

```
 '
 '
50 OPEN "A", #3, "FILE-A3"
 '
 '
100 WRITE#3,"SANYO","USER'S GUIDE"
110 CLOSE
120 END
```

CHAPTER 10 - FUNCTIONS

There are two kinds of functions, numeric functions (the function of which the value is numeric) and string functions (the function of which the value becomes a character string). Some functions have the parameter(s), which are described in a pair of parentheses "()" immediately after the function name. If the function has more than two parameters, each of them is delimited with a comma. String functions have "$" in their name.

Numeric function	Basic function	ATN, COS, EXP, LOG, SIN, SQR, TAN
	Numeric function	ABS, CDBL, CINT, CSNG, FIX, INT, SGN, RND
	String-numeric function	ASC, INSTR, LEN, VAL
	Disk function	CVD, CVI, CVS, DSKF, EOF, LOC, LOF
	Print function	SPC, TAB
	Others	CSRLIN, ERL, ERR, FN, FRE, INP, LPOS, POS, PEEK, USR, VARPTR, SEG
String function	String function	LEFT$, MID$, RIGHT$
	String-numeric function	CHR$, HEX$, OCT$, SPACE$, STR$, STRING$
	Disk function	ATTR$, MKD$, MKI$, MKS$, PACK$, UNPACK$
	Others	USRn, FN INPUT$

The function values and parameters of the numeric functions are integers, single-precision, real numbers or double-precision real numbers. The operation in the internal system of the basic function is carried out in double-precision, so that the precision of the function value of the basic function is improved if the parameter is specified in single-precision. In the explanation of each function, certain symbols are used to stand for the specific parameters. The examples in the right column of the table show practical uses of the function.

Explanatory Symbols	Meaning
X, Y	<numeric expression>
I, J	<numeric expression> of which evaluation value becomes an integer. (In case it becomes a real number, it is automatically rounded,counting the fractions five and over and disregarding the rest.)
X$, Y$	<string expression>

No.	Function Name and Parameter	Function Value	Example
1	ABS(X)	Absolute value of X	ABS(5*(-1))-->5
2	ASC(X$)	ASCII code conversion character in X$; an error occurs when X$ is the null string.	ASC("ABC")-->65
3	ATN(X)	Arctangent of X (trigonometric function) (Radian)	ATN(3.14159)--> 1.26263
4	ATTR$(a) a:<unit name>, #<file number, or <file name>	Attribute symbol of specified file (W, S or null string)	ATTR$(B:) ATTR$(#2) ATTR$("FILE")
5	CDBL(X)	Converts X to a double-precision real number	CDBL(1.234)
6	CHR$(I)	Returns the character having an ASCII code "I"	CHR$(65)-->"A"
7	CINT(X)	Converts X into an integer (counting fractions five and over and disregarding the rest); when X -32768 or X 32767, an error occurs.	CINT(2.5)-->3
8	COS(X)	Cosine of X (trigonometric function) (Parameter is in radians)	COS(3.14159)-->-1
9	CSNG(X)	Converts X into a sin-	CSNG(1.2345678)

		gle-precision real number	-->1.23457
10	CSRLIN	Vertical coordinate (line pointer) of the current Cursor position (1-25); POS function for horizontal coordinate (character pointer)	LOCATE POS, CSRLIN+5 (Moves Cursor to 5 lines lower)
11	CVD(8-byte characters)	Double-precision real number of 8-byte characters' numeric value (Inverse conversion of MKD$)	
12	CVI(2-byte characters)	Integer of 2-byte characters' numeric value	To be used for read-in operation
13	CVS(4-byte characters)	Real number of 4-byte characters' numeric value (Inverse conversion of MKS$)	For details, see items 36 to 38
14	DSKF(<unit name>)	Blank area of the disk stored in the unit specified by <unit name> (kilo bytes)	PRINT DSKF(B:) 128
15	EOF(<file number>)	When the file with the specified number reaches EOF (End of File), the value becomes -1; otherwise, 0 is assigned.	IF EOF(3) THEN 200 (When the File #3 reaches EOF, the control jumps to 200. Only valid for "I" mode sequential files.)
16	ERL	Number of the line where an error occurs	IF ERR=2 AND ERL=100 THEN GOTO 1000
17	ERR	Error code	
18	EXP(X)	eX	EXP(1) --> 2.71828

19	FIX(X)	Returns the integer of X. Equivalent to SGN (X)*INT(ABS(X)). The major difference between FIX and INT is that FIX does not return the next lower number for a negative X.	FIX(3.25) --> 3 FIX(-3.8) --> -3
20	FRE	Number of bytes remaining in memory (decimal number)	PRINT FRE
21	HEX$(X)	Hexidecimal value of X	PRINT HEX$(255) FF
22	INKEY$	Character to be keyed in; if no key is pressed, $ becomes null string.	10 IF INKEY$="A" THEN 1000 ELSE 10 (The control keeps waiting until "A" is keyed in. Once "A" is pressed, it jumps to 1000.)
23	INP(I)	Transfers 1 byte from I/O port of CPU. "I" indicates the number port.	If nothing is connected to the I/O port, the results are unpredictable
24	INPUT$ (I[, [#]J1%J])	Returns (I) characters input from the keyboard, the disk, or the serial ports. #J is the file number and %J is the port number. When only "I" is written, it is considered to be the keyboard. Input data is not displayed on the screen.	A$=INPUT$(5) X$=INPUT$(10,#2) Y$=INPUT$(1,%1)
25	INSTR ([I],X$,Y$)	Location of Y$ in X$; searching starts from Ith character	X$="ABCXXXDEFYY" Y$="XXX" INSTR(X$,Y$)-> 4
26	INT(X)	Returns the maximum integer not exceeding X	INT(258.4)-> 258 INT(-301.8)-> -302

27	LEFT$(X$,I)	Returns the leftmost I characters of X$; if I<0, or I>255, an error occurs.	A$="ABCDEF" LEFT$(A$,3)->"ABC"
28	LEN(X$)	Number of characters IN X$ string.	LEN("ABCDE") --> 5
29	LOC(<file number>)	Random file:the next record number to be used. Sequential file: number of records written or read since the file was opened.	LOC(1)
30	LOF(<file number>)	Number of records written or read in the specified disk file. (128 bytes/rec)	LOF(1)
31	LOG(X)	Natural logarithm of X	LOG(2)-->0.693147
32	LPOS	Position of printing head within the printer buffer	
33	MID$(X$,I[,J])	Returns J(number) of characters from X$ beginning with the Ith character. When J is omitted, returns the characters between the Ith and the end of X$	A$="1234567" MID$(A$,3,2)->"34" MID$(A$,5)->"567"
34	MKD$(<expression>)	<expression> becomes double-precision number CVD for inverse conversion	Converts numeric values to string values. MKI$ converts an integer to a 2-byte string. MKS$ converts a single precision

35	MKI$(<expression>)	<expression> becomes an integer. CVI for inverse conversion	string. MKD$ converts a double precision number to an 8-byte string Mainly used for writing out random files.
36	MKS$(<expression>)	<expression> becomes a real number. CVS for inverse conversion	
37	OCT$(I)	Character string of octal notation of I	OCT$(256)-->"400"
38	PACK$(X$)	X$ is a character string which indicates the numeric value consisting of 0,~9, +, -, ., D, E, and spaces. Creates a character string whose number of bytes is a half of X$'s. UNPACK$ is used for inverse conversion	LEN(PACK$ ("123456")) --> 3 Used for writing in the disk file.
39	PEEK(I)	Decimal data in memory at address "I". (numeric value)	POKE &HA000,&HFF PRINT PEEK(&HA000) 255
40	POS	Horizontal coordinate (character pointer) of the current Cursor position (1-80); CSRLIN function for vertical coordinate (line pointer)	
41	RIGHT$(X$,I)	Returns the rightmost characters of X$ to the Ith character	A$="ABCDEF" RIGHT$(A$,2)->"EF"

42	RND[(<arithmetic expression>)]	Returns a pseudo-random number in the interval 0 to 1 X<0 will generate the same number for a given X X>0 generates random numbers between 0 and 1 X=0 repeats the last number generated	
43	SGN(X)	Sign of X: SGN(X)=-1 X<0 SGN(X)=0 X=0 SGN(X)=1 X>0	
44	SIN(X)	Sine of X (trigonometric function in Radians)	SIN(3.14159/2)->1
45	SPACE$(I)	Returns a string of spaces of length I.	
46	SPC(I)	Outputs I(number) of spaces (used only in PRINT/LPRINT statement)	
47	SQR(X)	Square root of X	SQR(2) --> 1.41421
48	STR$(X)	Converts the numeric value "X" to the character string. VAL(X$) is used for the inverse conversion	STR$(100)-> " 100"
49	STRING$(I, J\|X$)	Returns a string of length I whose characters have the ASCII code J or the first character of X$	STRING$(3,"ABC") --> "AAA" STRING$(4,&H43) --> "CCCC"

50	TAB(I)	Tabulation spacing to print from the Ith character position counting from the left end. I = 1 - 80 Effective only in PRINT/ LPRINT statement. In PRINT statement, this function also works upon the leftward positions further than the current Cursor position.	PRINT TAB(20);"A"
51	TAN(X)	Tangent of X (trigonometric function in Radian)	TAN(3.14159/4)->1
52	UNPACK$(X$)	Restores the packed character string to the original status	Used for read-out operation from the disk files
53	VAL(X$)	Returns the numeric value of X$ character string. (Character numeric data conversion; STR$ is the opposite.	VAL("12345")->12345 VAL("123AAA56")->123
54	VARPTR (<variable>)	Address where <variable> is stored. As the numbers over 8000H become negative, it is better to convert the numbers using HEX$ function. (When the variable is a numeric value, the lower byte address is output. When the variable is a character date, the lower byte address of the character string index is output.	HEX$(VARPTR(A$))
	VARPTR (#<file number>)	Lowest byte address of the buffer area of the file having the specifield file number.	HEX$(VARPTR(#2))
55	FN<name>	User function defined in DEF FN statement.	

56	USR[<n>](X)	User function defined in DEF USR statement.	

CHAPTER 11 - PROGRAM EDITING

Line Edit Mode

(1) Entry of a Program

When the system displays in the BASIC Mode a Prompt > and a Cursor on the screen, (the cursor blinking at the position immediately after a Prompt) the system has initialized the BASIC program or returned to the BASIC Mode after execution of various instructions, the operator can perform the entry of a program line.

For the entry of a line, write the line number first, then the statement, and <CR> at the end. When <CR> key is pressed, the line is loaded in memory, and the system checks for Editing Errors (See the technical reference section, ERROR MESSAGES).

The operator deletes characters by pressing the DEL key until the desired character is reached. <CE> functions to delete the line entirely (In the AUTO Mode only the line number is left valid as entered even if <CE> is pressed. Otherwise, the line number is also deleted).

A complete program line is formed from the line number to <CR>. LF may be included in the line. When LF is keyed in, the lines are displayed or printed, however, in the program logic, one line continues from the line number to <CR>.

Line length can be up to 255 digits including a line number, space(s) and LF. The number of LF in one logical line can not exceed 19.

The program in memory is controlled with the line numbers, it can be listed or executed in the sequence of line numbers. A greater line number follows a smaller in memory with the program stored according to this order, so lines can be entered in any sequence. If the line already exists in memory, the line whose number is the same as the line entered will be stored as keyed in. When the lines of the program have filled memory or are greater than 65535, the system sends out the corresponding error message.

The MBC-4000 series program editor works in the line mode only. After a program line is entered and a <CR> is pressed, the line is stored in memory. If a change is to be made, the EDIT command is used in the following way:

To edit a certain line in a program type EDIT then the line number. The line appears on the screen with the cursor one line below. By using the --> cursor right key the old line characters are repeated up to the change to be made. the DELETE key is pressed to delete the desired characters. The <-- cursor left key erases the edited line and puts the prompt at the beginning again. All new characters entered are inserted into the old line at the point where the prompt character is. The RETURN key leaves the EDIT mode and enters the new line as changed. The BREAK key leaves the EDIT mode and the old line

unchanged.

(2) Deletion of a program

The following two instructions carry out the delete operations of a program.

a. Deletion of the entire program. . . NEW statement

b. Deletion of a part of the program . . . DELETE statement

When the operator wants to delete a particular line only, its line number and <CR> can be keyed in, erasing that line from memory.

(3) Listing the program

While control stays in the BASIC Mode, the program which is stored in memory can be displayed or printed out entirely or partly.

a. Listing of the program on the screen. . . LIST statement

b. Listing of the program to the printer. . . LLIST statement

When the operator wants to stop the listing temporarily, the space bar of the keyboard is pressed. The listing operation is halted immediately. When any key is pressed again, the listing operation continues. With this operation, the user can confirm the contents of a program regardless of its length. If the user wants to discontinue the listing operation while it is in the halt state or in process, the BREAK key is pressed. The system then stops the listing operation and returns the control to the BASIC Mode.

(4) Saving the Program

The program in memory can be saved onto the disk files by use of the SAVE statement. The program transferred onto the disk file can be read into memory by use of the LOAD statement.

(5) Merging of Two Programs

With the MERGE statement, the program in memory can be merged with another program which is read in from the disk. In the BASIC Mode the operator keys in the MERGE statement with the following format:

MERGE "PRGM-B" <CR>

The system reads in "PRGM-B" from the disk and makes up a new program merging it with the existing program in memory. When the same line number of the program is included in "PRGM-B", the contents of the line are changed as written in "PRGM-B".

(6) AUTOmatic and RENUMbering Operations

When keying in, the system can create and display the line numbers automatically. the AUTO command is used for this purpose.

The operator assigns the starting line number and the increment to the parameters of the AUTO command. The system automatically displays the succeeding line number every time a line has been keyed followed by a <CR>. (In the AUTO Mode the line number is not deleted even though the other part of the line is deleted by use of CE key).

To stop the AUTO operation, BREAK key is pressed.

For overall line number modification the RENUM statement is used. With this statement, the operator can change all the existing line numbers to make room for additional lines or to make the program easier to read.

CHAPTER 12 - DISK FILES

Types of Files

The following disk files are handled under SANYO-BASIC.

- Disk files
  - Program files
  - Data files
    - Sequential files
    - Random files
    - Index sequential files

Every file has a name with which a Call operation can be used.

File Names

A name is assigned to the particular file in advance. The specification of a file name is done according to the rules mentioned below:

a. A file name consists of 11 characters (8 characters for file identification and 3 characters for extension). A period (.) is placed to delimit the two parts.

```
--------  .  ---
filename     extended file name (qualifier)
```

Example:
```
TEST.BAS
TESTPRGM.BAS
DATAFILE.A1
FILE.ABC
BASIC.CMD
```

The extended file name is not necessary in a call statement. When more than 8 characters are allocated for a file name, the characters between the 9th and 11th are regarded as an extended file name. the 12th and over are disregarded.

Example: TESTPROGRAM-A --> TESTPROG.RAM

The extended file name indicates the particular group to which the file belongs. As "CMD" is used as the extended file name of a system program and "BAS" as that of a BASIC program, the user is advised to avoid the use of these extended file names.

When the following instruction words are used in the BASIC Mode, and the user does not write the extended file names, the system automatically adds "BAS" to the file names.

- o Initialization of BASIC (Refer to Chapter 9)
- o CHAIN
- o LOAD
- o MERGE
- o RUN
- o SAVE

b. The file name must begin with a letter or $. For the second and later characters, any of the alphanumerics or the following symbols can be used.

#, $, %, &, ', (,), +, -, /

Both uppercase and lowercase letters are available, so the user must distinguish between the two.

c. Any character type (constant, variable or string) may be used in writing a file name in the program.

```
Example:  (1)  SAVE "PRGM-A"          ⟶  PRGM-A.BAS
          (2)  LOAD "PRGM-A"          ⟶  PRGM-A.BAS
          (3)  MERGE "PROGRAM-A"      ⟶  PROGRAM-.A_ _
          (4)  A$="FILENAME"
               OPEN "O", #1, A$           FILENAME._ _ _
                             ↑
                             └──a file name

          (5)  A$="FILE-"
               I=3
               OPEN "I", #I,A$+RIGHT$(STR$(I),1)  ⟶  FILE-3_ _ _
                             └──────────┬──────┘
                                        └──Filename

          (6)  OPEN "R", #3, "FILE/4.ACC"  ⟶  FILE/4.ACC
```

d. When writing a file name, the specification of a drive unit can be also written.

```
LOAD "B:PRGM-A"          (To specify that PRGM-A is loaded from
                         B drive unit.)
```

The drive unit to be used is specified with either A, B, C, or D followed by a colon (:) as a delimiter. When optional units are installed; B, C, or D units become valid. If there is no unit specification in a file name, A is assumed.

e. In FILES, LFILES, KILL, and SET statements, an asterisk (*) can represent all the file names or extended file names, and a question mark (?) represents any of the characters.

Example:

FILES "B:*.*" or FILES "B: *	Indicates all the disk files in B drive unit.
FILES "*.BAS"	Indicates all the files having "BAS" in their extended file names.
KILL "TEST??.*"	Indicates that every file whose name is of six characters beginning with TEST should be erased.
SET"*.DAT",W	Indicates that the write protect is set on every file whose extended file name is DAT.
KILL "A????.BAS"	Indicates that every file whose file name is of five characters beginning with A and has "BAS" extended file name should be erased.

Number of Files

Up to 15 data files can be opened at the same time in a BASIC program. When any file is used, a buffer area is automatically reserved in memory for write/read operations to disk.
The size of the buffer area becomes greater as the number of data files to be opened also increases. The user must specify the number of files previously in the initialization of the BASIC program so that memory is used efficiently.

As a 300-byte buffer area is reserved for one open file, 4.5K bytes are requrired if the user intends to use 15 files, which influences other areas in memory. In forming a program, the volume of the program as well as the amount of data should be considered to determine the number of files.

The number of bytes available in memory is displayed at the time of initialization of the BASIC program is the capacity after the buffer area is reserved.

Instructions for Files

The disk files are:

- Listed by use of the FILES/LFILES statement.
- Erased by use of the KILL statement.
- Renamed by the NAME statement.

They can be provided with two kinds of attributes--WRITE PROTECT to prohibit the writing of data on the disk. The file provided with the SYSTEM attribute cannot be listed out by the FILES/LFILES statement. And the system cannot be written into or erased by the KILL statement as the file is provided with the WRITE PROTECT attribute. For details see Chapter 9 (the respective sections of "INSTRUCTION WORDS") and

Program Files

A program file is created to load the stored program into the disk file by use of the SAVE statement. The program file is read into memory by the LOAD statement or merged with the existing program in memory by the MERGE statement. It is then executed by the RUN command. When the program file names are specified in the SAVE, LOAD, RUN, MERGE, or CHAIN statement or used for initialization of the BASIC Mode, "BAS" is added automatically as an extended file name even if the user omits the particular extended file name.

Data Files

The data files are classified into two types--sequential files and random files, and are the disk file types where some amount of data can be loaded. The data is created by the user and written on the diskette and can be read out by a BASIC program.

To write the data into a file or to read the data out of a file, the user must open the corresponding file with the OPEN statement and close it by the CLOSE statement after it has been used. However, when the END, LOAD, RESET, SYSTEM, DELETE, or MERGE instructions are executed, all the open files are closed automatically.

Sequential File

A sequential file is a file that is ordered sequentially. The data can only be written or read from the begining. It neither can be written nor read from the middle of the file.

1. Writing into the file

a. Open the file with "0" mode .	OPEN "0",.....
b. Write the data into the file.	PRINT # USING.. WRITE #.....
C. Close the file.	CLOSE#.....

To add the data to the file already having data, specify OPEN "A",... to open, then the data can be added to the end of the existing data of the sequential file.

2. Reading from the file.

a. Open the file with the "I" mode.	OPEN "I",.....
b. Read the data from the file.	INPUT #..... LINE INPUT#....
C. Close the file.	CLOSE #.....

Example

```
10 OPEN "O", #1, "A-FILE.DAT"
20 INPUT "CODE";C$
30 IF c4="0000" THEN 90
40 INPUT "FIGURE" ;N%
50 INPUT "MONEY"; A
60 WRITE#1, C$, N%, A
70 PRINT
80 GOTO 20
90 CLOSE

100 OPEN "I", #2, "A-FILE.DAT"
110 IF EOF(2) THEN 150
120 INPUT#2, K$, S%, K
130 PRINT K$; TAB(10); S%; TAB(30); K
140 GOTO 110
150 CLOSE#2
```

Random Files

With a random file the write-in or read-out of data starts from any part of the file. This functions effectively for random processing of data. The random file can be used with the following procedure.

(1)	Open the file in "R" mode. Specifications of record length are required. If omitted, the default value "128 bytes" is assigned for one record.	`OPEN "R", #1, "R-DATA",A`
(2)	Define the random buffer area.	`FILE#1, 8 AS A$,32 AS B$`
(3)	Enter the data into the random buffer area by the LSET/RSET statement. The numeric data must be converted to string data in advance.	`LSET A$=MKD$(D#)` `LSET B$=X$`
(4)	Write the data from the buffer area into the disk file with the PUT statement. Specify the write-in position with the record number.	`PUT#1,N%`

(5)	Read the data from the disk file to the buffer area with the GET statement. The numeric data must be converted from the string to the numeric value.	`GET#1, M%`
		`D#=CVD(A$)`
		`PRINT M%, B$, D#`

(6)	Close the file	`CLOSE#1`

Up to 32767 records can be written in a random file. The write-in and read-out modes are not specified for a random file. One random file can be used in either mode if necessary, after the OPEN and FIELD statements are executed.

If a record that has not been written is read out, the contents of the record are unpredictable.

Example

```
10 OPEN "R", #2, "R-FILE", 50
 20 FIELD #2, 2 AS C$, 18 AS N$, 30 AS M$
 30 INPUT "CODE"; C%
 40 IF C% = 0 THEN 500
 50 INPUT "NAME"; NA$
 60 INPUT "DATA";DA$
 70 RSET C$=MKI$ (C%)
 80 LSET N$=NA$
 90 LSET M$=DA$
100 PUT#2, C%
110 GOTO 30
 .
 .
500 INPUT "CODE";N%: IF N%=9999 THEN 700
530 GET #2, N%
540 PRINT TAB(10);CVI(C$);TAB(20);N$;TAB(50);M$
550 PRINT
560 GOTO 500
 .
700 CLOSE #2
 .
 .
```

When the EOF mark is written at the end of a random file, CHR$(&H1A) must be added at the beginning of the file next to the records which are used by the user (In case of SORT, etc.).

Index Sequential File

The data can be written and read anywhere in the index sequential file using the key. If the key is not specified, the data can be processed sequentially by the ascending key order.

The procedure is explained as follows:

(1) Allocate the file area on the disk.
```
CREAS "ISF.MAS", 15,1,30,10 CREATE
```

(2) Open the file with "IS" mode.
```
OPEN "IS", #1, "ISF.MAS"
```

(3) Define the buffer. Be careful about the key position and the length when delimiter is used in the FIELD statement.
```
FIELD #1,3 AS CODEIS $, 12 AS NAMEIS $
```

(4) Stores the data in the buffer using the LSET/RSET instructions. (Write)
```
LSET CODEIS $=CODE $
LSET NAMEIS 4=NAME $
```

(5)' Write the data from the buffer into the disk using the PUT instruction. (Write)
```
PUT #1, CODEIS $
```

(4)' Store the data from the disk to the buffer using the GET instruction with the key. (Random read)
```
GET #1, "A02"
```

(4)' Specify the record pointer using the SET instruction. (Sequential read)
```
SETKEY #1, "A05"
```

(5)" Read the data from the disk into the buffer using the GET instruction. (Sequential read)
```
GET #1
```

(6) Delete the record using the KILL instruction.

(7) Close the file.

(Example)

```
 100 CLS
 110 '
 120 PRINT "***INDEX SEQUENTIAL FILE CREATE***"
 130 '
 140 CREATE "ISF.MAS",15, 1, 3, 10
 150 OPEN "IS",#1, "ISF.MAS"
 160 FIELD #1, 3 AS CODEIS$, 12 AS NAMEIS$
 170 FOR I=1 TO 10
 180 READ CODE$,NAME$
 190 LSET CODEIS$=CODE$
 200 LSET NAMEIS$=NAME$
 210 PUT #1, CODEIS$
 220 PRINT CODEIS$, NAMEIS$
 230 NEXT I
 240 PRINT
 250 '
 300 PRINT "******* RANDOM GET *******"
 310 '
 320 GET #1, "A02"
 330 PRINT CODEIS$, NAMEIS$
 340 GET #1, "C01"
 350 PRINT CODEIS$, NAMEIS$
 360 GET #1, "B05"
 370 PRINT CODEIS$, NAMEIS$
 380 GET #1, "D01"
 390 PRINT CODEIS$, NAMEIS$
 400 PRINT
 410 '
 500 PRINT "******* SEQUENTIAL GET (KEY) *******"
 510 '
 520 SETKEY #1, "A05"
 530 FOR I=1, TO 6
 540 GET #1
 550 PRINT CODEIS$, NAMEIS$
 560 NEXT I
 590 '
 600 PRINT "******* KILL (KEY) *******"
 610 '
 620 KILL #1, "A01"
 630 SETKEY #1
 640 FOR I=1 TO 9
 650 GET #1
 660 PRINT CODEIS$, NAMEIS$
 670 NEXT I
 680 CLOSE #1
 690 END
```

```
700 '
710 '******* INDEX DATA *******
720 '
730 DATA C01, CUBA
740 DATA C02, CANADA
750 DATA D02, DENMARK
760 DATA A05, ANKARA
770 DATA A01, AMSTERDAM
780 DATA D01, DETROIT
790 DATA A02, ARABIA
800 DATA A03, ARGENTINA
810 DATA B08, BURMA
820 DATA B05, BRAZIL
```

Input/Output of Files and Disk Images

(1) Sequential files

The PRINT#, PRINT# USING and WRITE# statements are used for the write-in operation of the sequential file. The disk images of the sequential file data depends on these respective statements.

```
A$="RADIO"
B$="CASSETTE"
C$="12345"
                              Disk image
1.  PRINT#1, A$;B$;C$;        RADIOCASSETTE12345(CR)
2.  WRITE#1, A$;B$;C$;        "RADIO", "CASSETTE", "12345"CR
3.  INPUT#1, A$,B$,C$
```

When the WRITE#1 file is used

Then, A$="RADIO"

When the PRINT# file is read

Then, A$="RADIOCASSETTE12345"

Using PRINT#1 the data written in #1. becomes one data item. Therefore, when using the PRINT# statement to write the data to the disk, the user should key in:

```
4.  PRINT#1, A$;",";B$;",";C$
```

So that the three data items are read separately by the INPUT# instruction. The WRITE# statement automatically encloses each data item in quotation marks (" ") and places commas (,) between the items. When the LINE INPUT# statement is used, the line beginning with WRITE# and terminating with CR is read as one data item.

The PRINT# statement writes the numeric data to the disk file while the PRINT statement displays it on the CRT screen.

```
IF:     A=80.5
        B=13.15
```

5. When PRINT#1, A, B is keyed in, unnecessary TAB space is placed between A and B; therefore,

```
6.  PRINT#1, A; ","; B should be keyed in.
```

(2) Random files

As the random file defines the buffer area to be a string field by use of the FIELD statement, the data to be written in must be of string type. The following is specified as the string data:

Integer	MKI$(I)	(To be 2-bytes string)
Real number	MKS$(R)	(To be 4-bytes string)
Double-precision real number	MKD$(D)	(To be 8-bytes string

These string data items are written in the buffer area by the LSET/RSET statement and output by the PUT statement. In reading them out, they are converted again to numeric data through the use of CVI (for integer), CVS (for real number) or CVD (for double-precision real number) function. The string data are unchanged and transferred to the buffer area for input.

(3) Packed data

One character of the string data is written in one byte, however, the packed data has two characters in one byte for effective use of the disk. Only 16 types of characters can be packed - they are the numbers from 0 to 9, +, -, . (period), D, E, and a space. This means that only the numeric characters are permitted for packed data. The packed data is created by the PACK$ function and when reading it out, it is restored by the UNPACK$ function.

```
A$="1234.567"        8 bytes
P$=PACK$(A$)         4 bytes
```

When P$ is written on the disk, the number of bytes becomes half of A$. To read out the data, key in as

```
A$=UNPACK$(P$)
```

and the data is restored as

```
A$="1234.567"
```

The allowable number of files on one diskette including the system program files of which extension file names have ".CMD", SANYO-BASIC program files, and all the data files, is determined as follows:

	Allowable file number on one diskette
MBC- 4000 series 5" diskettes	128

Note: As the file number is counted by the extent in the operating system, it may be less than the above value depending on the size of the data. For detail refer to the explanation of STAT in Chapter 6.

CHAPTER 13 - DISK UTILITY PROGRAMS AND THEIR USE

Introduction

Your SANYO series computer comes with utility programs that will allow you to fully use the system in a plurality of ways. This chapter will present these in an indexed form for your use. In using these programs follow the instructions with great care as data may be lost with a mis-entry. Most of the files are under the CP/M operating system. Refer to Chapter 6 first if you are not familiar with the workings of CP/M file structures.

Copying Disks

It is always a good habit to copy master diskettes and diskettes with important data on them. This backup avoids problems that may arise if you mistakenly destroy programs or data on a diskette.

Single Drive

Step 1 If you have a single drive, you will use the Filecopy Utility. When the system input symbol is displayed (A>), enter FILECOPY. The following prompt will be displayed:

```
SINGLE DRIVE FILECOPY VER-X.X
ENTER FILE NAME (OR DEPRESS <CRTL-C> TO REBOOT)
```

Step 2 Enter the file name to be copied, then depress the RETURN Key. The following prompt will be displayed:

```
INSERT SOURCE DISK THEN PRESS <RETURN>
```

Step 3 Place the copy disk which contains the file you wish to copy in the drive and depress the RETURN key. The following prompt will be displayed:

```
INSERT DESTINATION DISK THEN PRESS <RETURN>
```

Step 4 Place the disk on which the file is copied in the drive and depress the return key. When the file is copied, the following prompt will be displayed:

```
FILE COPY COMPLETED
ENTER FILE NAME (OR DEPRESS <CTRL-C> TO REBOOT)
```

Depress the CTRL-C keys and you will be placed back into the DOS mode providing the disk in the drive has CP/M on it. If it doesn't, insert the original disk or a CP/M disk in the drive before the CRTL-C keys are depressed.

Dual Drives

If you have two drives you may use the COPYDISK Utility. This disk copy utility allows you to copy the entire disk in one command. Depress the RETURN Key after each entry.

When the system input symbol (A>) is displayed, enter COPYDISK. The following prompt and instructions are displayed:

```
CP/M-86 Full Disk COPY Utility Version 2.0
Enter Source Disk Drive (A-D)?
Destination Disk Drive (A-D)?
```

Enter the source drive name (the name of the drive the disk is on that you want copied). The destination disk drive contains the blank disk.

The following prompt is displayed after you enter the drive names:

```
Copying Disk A: to Disk B:
Is this what you want to do (Y/N)?
```

You have the opportunity of exiting form the disk copy utility if you enter an N. If you wish to continue, enter Y .

The following prompts appear:

```
Copy started
Reading Track XXX
Writing Track XXX
Verifying Track XXX
```

These last three prompts will appear until the entire disk is copied.

You may copy another disk when the disk copying instructions are displayed again.

File Transfer Program

The transfer of files can occur between the SANYO computers with the following program utility. In this way files based on 8" diskettes can be translated to 5 1/4" diskettes and vice versa.

The transfer will be made through the RS-232-C ports of both machines. This means a cable harness must be made (following chart). The transfer parameters have been set to be

LINE 1 PORT FOR BOTH MACHINES
4800 BAUD
8 Bits
No Parity Checked
128 Bytes per block transfer unit

Cable to be Used Between the Two Computers

LINE 1 PORT TERM	PIN	LINE 1 PORT PIN	TERM
SD	2	2	SD
RD	3	3	RD
RTS	4	4	RTS
CTS	5	5	CTS
SGRD	7	7	SGRD

FILE NAMES FOR THE UTILITY PROGRAMS AS STORED ON DISKETTE

SENDING SIDE MTRN.CMD

RECEIVING SIDE FRCV.CMD

THE FOLLOWING PROCEDURE SHOULD BE USED.....

1. Turn on power to both units and connect the special cable assembly to the RS232 port of both computers.

2. The RECEIVING UNIT must now have its proper transfer file program loaded. Use the chart above and the CP/M DIR command to access the file. Load it.

   A> MRCV

3. The SENDING UNIT must now have its proper transfer file program loaded followed by the filename you wish to transfer. Use the chart above and the CP/M DIR command to access the file. Load it along with the filename for the transfer. The RECEIVING UNIT MUST have already been loaded with the Receiving program.

   A> MTRN {File name}

   CP/M PIP protocols are allowed.

4. The RECEIVING UNIT will then respond with the the following message:

   SELECT DRIVE (A,B,C or D)?

   Key in the Drive name for the Receiving side.

5. File transfer will instigate with a message SUCCESS at the end of each file transfer. At the end of the multiple file transfer (If you had specied multiple transfers with the *,? parameters of CP/M) the displays are:

```
Sending Side:      TRANSMIT COMPLETE
Receiving Side:    RECEIVE COMPLETE
```

6. Should you get an error, LINE ERR displays.

   Depress CTRL/RESET to initialize both the sending and receiving computers and restart at the top of the procedure.

Procedure to Format a Diskette

Diskettes must be "formatted" or placed into a known state prior having records placed onto them. There is an easy way to accomplish this. The system disk carries a routine called FORMAT to help you accomplish this. You may also wish to refer to Chapter 6 which in the introductory text for CP/M.

Assume in the following that the source disk is file A and that the destination disk is drive B. Drive B will be used as the formatting drive. Follow the procedures above for single drive parameters.

1. Place Standard System Diskette into drive A. Place New (unformatted) diskette into drive B.

2. "COLD START" by depressing CTRL and RESET simultaneously.

3. The screen will read the operating system from drive A and will respond with the prompt:

   ```
   A>
   ```

4. Key in the letters FORMAT after the prompt and press return (CR or RETURN).

5. The program will be read and the screen shows;

   ```
   FORMAT
   DRIVE NAME TO BE FORMALIZED
   (OR RETURN TO EXIT)__
   ```

6. Key in B which is the drive to Format. Hit CR or RETURN.

7. The system will echo the response and then start the format process. After the finish of execution you will see:

   ```
   FUNCTION COMPLETE
   DRIVE NAME TO BE FORMALIZED
   (OR RETURN TO EXIT)__
   ```

   This is so that multiple diskettes can be formatted. If you are through simply hit CR

Procedure to Copy the Operating System

When copying files from one disk to another you should realize that the operating system is not transferred. To copy this system one must use the following utility. See also Chapter 6, SYSGEN.
This utility will transfer the operating system (auto boot/ auto load) from the source file in drive A to the destination file in drive B.

1. Perform a COLD START sequence or a WARM START (CONTROL+RESET or CONTROL+C).
   Drive A should contain the diskette that has the operating system on it.
   Drive B contains the destination diskette.
2. After the prompt message key in SYSGEN ,

```
A> SYSGEN  (CR)

SYSGEN VER-Cx.x

ENTER SOURCE DRIVE NAME (OR PRESS RETURN KEY TO SKIP). A

At this point make sure the source system
disk is in drive A.

SOURCE ON A, THEN PRESS RETURN KEY (CR)

ENTER DESTINATION DRIVE NAME (OR PRESS RETURN KEY TO REBOOT).B

Make sure your new disk is on drive B

DESTINATION ON B, THEN PRESS RETURN KEY  (CR)

FUNCTION COMPLETED

ENTER DESTINATION DRIVE NAME (OR PRESS RETURN KEY TO REBOOT).
```

If you have one drive, enter A for all drive prompts.
Remember to take out the system disk when the destination prompt is displayed and insert the new disk.

Procedure for Copying All the Files from A to B

Here we present the technique (under CP/M) to copy all the files from the source diskette in drive A to the destination diskette in drive B. The files will be transferred under the same name that they had before.

We will use the PIP command. For further details on PIP refer to the CP/M chapter of this manual which is Chapter 6.

1. Place the diskette into drive A that has the operating system.

2. Perform a WARM or COLD START (CTRL + C or CTRL + RESET).

3. After the CP/M prompt type PIP B:=*.* and depress (CR) or RETURN.

```
A> PIP B:=*.*    (CR)
```

   NOTE: If you are in PIP mode a "*" prompt will appear. In this mode follow the "*" prompt with the same message:

```
* B:=*.*    (CR)
```

4. The system will annotate a COPYING message and will then list all the files it is copying from source drive A to destination B.

Sort Utility

Your SANYO computer has the ability to sort files. Sorting of files allows one to selectively establish the sequence of string or data items. This special utility, having the filename SORT compares the records with fixed length in a file according to the designated key items or "mates" and reorders them into a new created file in a specified order (ascending/descending). The parameters for the Sort are inputted in one of the following two ways:

1. A key in sequence at Sort execution.
2. From a parameter file. This file is made by SANYO-BASIC or SORT, the called file routine.

SEQUENCE OF OPERATIONS

During SORT execution the following takes place:

1. SORT key items are taken out of the record and a two byte index number (0 to 65K) is affixed to each.

2. Each key item is placed into a work area in main memory. This is a field of 36K bytes. If we exceed this work area the disk (B) is used. For this operation we must have enough capacity left on the diskette. The formula is:

CAPACITY=((key length+2) x record number) + OFS + INT (mxrecord#/n)

OFS = Original File Size
m = 128-n(key length+2)
n = INT(128/(key length+2))

Be careful in chosing file names and extensions. Temporary files usually have he same filename but a different extension. These sorts of files may be destroyed as the SORT routine is constantly looking for more work area space. If the file having the same filename exists on the disk it will be destroyed. The allowable file numbers on one disk, including any temporary files is 64.

There are limitations on the SORT utility.

1. The maximum record length is 384 bytes per.
2. The maximum key items of a record is 20.
3. Maximum SORT key items is 5, and the total of key items should be within 124 bytes.
4. Maximum records of one file is 65,534 as this is the maximum resolution of the two byte index field.
5. At the end of every file a special character, END-OF-FILE (Hex value 1A) is required. In SBASIC sequential files EOF is automatically added. In random files EOF must be written by the user using a special program. An example follows for such a program.

```
1       OPEN "R",#1,"DATA",10
2       FIELD #1,5 AS A$, 5AS B$
.
.
.
10      LSET A$=CHR$(&H1A)  {this is EOF}
11      PUT#1,N  {N=final record number + 1}
12      CLOSE
```

When data is prepared use the PRINT USING format in SANYO-BASIC to fix the data length before sorting files.

EXAMPLE FOR ILLUSTRATION IN SUMMARY:

In summary then the SORT will first take a inputted file name, for example DATA.IN and will sort in the main memory into sub-allocations:

DATA.IN = DATA.AOA DATA.AOB DATA.AOC DATA.AOD

It will then MERGE members of the files (KEY FILE MERGE)

DATA.COA=(DAT.AOA)m(DAT.AOB) DATA.COB=(DAT.AOC)m(DAT.AOD)

Then a comparison of the last two files is made, the records of the input file are read at random, and they are written to an output file in a sorted order;

DATA.IN {+} COMPARE-KEY yields DATA.OUT to outputted file.

USE OF SORT BY EXAMPLE:

We can invoke and use the SORT utility from the CP/M system. At this point let's assume we have files on the diskette that have been established either through CP/M or SANYO-BASIC.

1. Key in SORT after the CP/M prompt.....
   A> SORT (CR)

2. The system will respond with;
   INPUT FNAME...
   Here enter the filename for the input file to be sorted.
   INPUT FNAME...B:DATA.NUM (CR)

3. The system will respond with
   FILE MODE...
   Here enter an R if the form of the input file was random or S for sequential files. Sequential files have class codes CR and LF at the end of a record.
   FILE MODE...R

4. The system will respond with

   RECORD FORMAT...
   This will designate the format that makes up one record of an input file. For sequential files it is the net byte number without the terminating CR and LF characters.

AS AN EXAMPLE.....

Assume in SBASIC you have made some string files as:
A$="ABCD"
B$="1234"
C$="EFGH"
D$="5678"
And have used the PRINT #1 as:
PRINT#1 A$;B$ and
PRINT#1 C$;D$
This will make the records on the file as;
A B C D 1 2 3 4 CR LF
E F G H 5 6 7 8 CR LF
The net byte number is 8 each. This is what must be entered. Also we must enter the number of pack (Pn) bytes, and the type of data:

B2 for integer 2 bytes
B4 for single precision real numbers 4 bytes
B8 for double precision real numbers 8 bytes

We must also enter a terminator 69 character (if so used)

RECORD FORMAT...20,5,P2,B4,69 (CR)
Since this is a NUM record have 5 bytes/item, they are single precision real numbers, and the record size = 100 bytes.

5. The system will respond with
RECORD SIZE...100 BYTES ITEM...5
SAMPLING DATA...

Here we designate the record sort mode in the input file. If you enter the character * the entire record of the input file will be sorted. You may specify a comparison expression if you wish. These take the form:

{"xxx" < (=)} In {< (=) "yyy"}

called the ANDed form.

Or this form:

In = "xxx"{,"yyy"}
called the ORed form.

In the above xxx and yyy are filenames limited to characters. Pn and Bn cannot be specified.

Up to 3 comparison expressions can be used, separated by ";".

EXAMPLES:

```
"ABC"<=I1<"XYZ"; I2="AA","BB"
          "I2<"AA"; I3="K"; "999"<=I4
```

6. The system will respond with

SORT SEQUENCE...

Here we specify the Sort key within 124 bytes. Three forms are possible:

In	Where n is the number of characters specifies an ascending sort.
U/In	The same ascending sort as above.
D/In	A descending sort.

EXAMPLE:

SORT SEQUENCE...I2,D/I3,D/I4
follow with a (CR) or RETURN

7. The system will respond with

OUTPUT FNAME...

Here specify the output file that will be the sort result. If you skip this the output file will have the same name as the input file. If you input a file name that exists on the diskette the contents of the old file will be destroyed as the new file is written.

EXAMPLE:

OUTPUT FNAME...B:DATA.OUT

After input perform a (CR)....

8. The system will respond with

```
PARAMETER FILE MAKE?...
        Here you must answer  Y  or  N
        to indicate your desire to make
        and register a parameter file

        If you specify  Y  for Yes a
        prompt will ask you for the
        name of the parameter file.
        (PARAMETER FNAME?...)
        In any case a (CR) or RETURN.
```

9. The system will respond with

```
SORT START OK?
        Here 3 answers are possible:

        Y...Yes, start the process.
        N...No, return to operating
                system mode.
        P...Enable parameter reinput.

        Y responses yield a response
        that starts the sort and ends
        with:

INPUT-DATA xxx RECORDS
END OF SORT RESULT IS {file name}
```

10. Let's look an example of a parameter file sort.

```
A> SORT B:PFILE
   SORT Ver 1.1
   INPUT FNAME...B:DATA.NUM
   FILE MODE...R
   RECORD FORMAT...20,5,P2,B4,69
   SAMPLING DATA...*
   SORT SEQUENCE...I2,D/I3,D/I4
   OUTPUT FNAME...B:DATA.OUT
   RECORD-SIZE...100 BYTES ITEM...5
   INPUT-DATA...200 RECORDS
   END OF SORT RESULTS IS...B:DATA.OUT

A>
```

A parameter file can have the form of a SANYO-BASIC comment file. It can be formatted in SANYO-BASIC nomenclature. The file called B:PFILE above has, for example, the following form:

```
1 'D:DATA.NUM        (CR)+LF
2 'R                 (CR)+LF
```

```
3 '20,5,P2,B4,69     (CR)+LF
4 '*                 (CR)+LF
5 'I2,D/I3,D/I4      (CR)+LF
6 'B:DATA.OUT        (CR)+LF
```

When NON is used as a extension name of a parameter file the contents of the parameter file will not be displayed during the sort execution.

Parameter files can be 115 byte

ERROR MESSAGES FOR THE SORT UTILITY

When inputting parameters you may see:

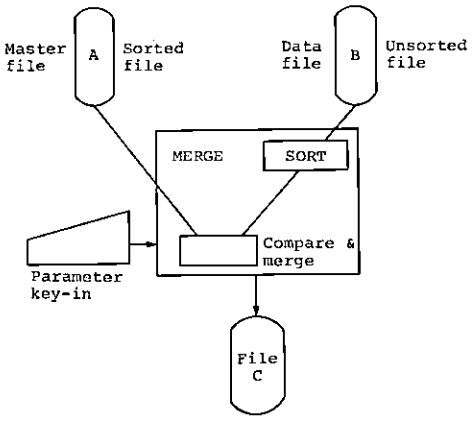
MISS FORMAT	A format other than specified is inputted (the wrong type).
RECORD FORMAT LENGTH OVER	Record has exceeded 384 bytes.
OVER KEY	More than 6 key items have been designated.
KEY FORMAT LENGTH OVER	The total of key items exceeds the 124 byte limit.

You may also see the following errors:

INPUT FILE NOT PRESENT	Designated file doesn't exist.
PARAMETER FILE NOT PRESESNT	Parameter file doesn't exist.
INPUT-FNAME ERROR	Errors are in the input file of the designated parameter file or file descriptor.
FILE-MODE-ERROR	Mode errors exist in the designated parameter file.
RECORD-FORMAT-ERROR	Errors exist is the record format of the designated parameter file.
SAMPLING-DATA-ERROR	Errors exist in the sampling data of the designated parmeter file.
SORT-SEQUENCE-ERROR	Errors exist in the sort sequence of the designated parameter file.
OUT-FNAME-ERROR	Errors exist in the output file of a specified parameter file.

Merge Utility

Your SANYO computer has the ability to MERGE files. This is usually done after a SORT routine has been performed on data files according to the designated key items. The MERGE routine combines the data file with a master file and outputs a newly created MERGE file.



The MERGE utility has certain limitations which are listed below:

1. DATA File length cannot exceed 36K-Bytes.....
   File length is defined as follows:

   (Length of all key items + 2) X (Number of Records)
2. The number of records cannot exceed 65,534.
3. The maximum record length is 384 bytes.
4. The maximum item numbers in one record is 20.
5. The maximum number of SORT key items is 5.
6. File End (EOF) designator is ASCII 1A (Hex).

The MERGE utility is invoked from the CP/M system prompt. There are two forms as in the SORT utility discussed in the previous section....

You may wish to key in parameters:

```
A> MERGE
```

You may wish to use a parameter file format:

```
A> MERGE {parameter file name}
```

The following sequence will occur after the request for MERGE. Let's assume we will be inputting keyed-in parameters:

1. The system will respond with

```
MERGE Ver 1.1
MASTER FNAME...
        Here input your master file
        EXAMPLE:
MASTER FNAME...B:MASTER.FL
```

2. The system will respond with

```
IN-DATA FNAME...
        Here input your input file
```

EXAMPLE:
IN-DATA FNAME...B:DATA.999

3. The system will respond with
FILE MODE.....

Here you must designate the form for the master and input files. An "S" indicates a sequential file and an "R" indicates a random file.
EXAMPLE:
FILE MODE.....R

4. The system will respond with
RECORD FORMAT...
Here you define the format of the byte numbers for both the master and input files. n defines character item byte numbers. Pn defines the number of numerical items. A Bn format is used to specify
B2 Integers
B4 Single Precision real #s.
B8 Double Precision real #s.

EXAMPLE:
RECORD FORMAT...20,5,P2,B4,69
Here 69 is the terminator.

5. The system will respond with (for our example)
RECORD SIZE...100 BYTES ITEM...5
and will then display:

SORT SEQUENCE
Here you must designate the SORT sequence within 124 bytes. The formats here are identical to those in the previous SORT section.

U/In Ascending
D/In Descending
n Item number

EXAMPLE:

SORT SEQUENCE...I12,D/I3,D/I4
After you enter and (CR)

6. The system will respond with
SORTKEY ITEM...3 (for our example)

And then will display:

OUTPUT FNAME...

Here you input the designated output file for the MERGE. If you omit the OUTPUT FNAME MERGED data is added to the Master File. If the inputed filename is already on the diskette that file will be destroyed.

After inputting the name depress (CR) and...

7. A system response will follow that states

MERGE START OK?

Here a response can be
a. Y or YES, start MERGE
b. N or NO, return to the Operating system.
c. P set up parameters mode.

Follow all responses with (CR).

8. The system responds with

PARAMETER FILE MAKE?

Here the system asks for definition for the registration file name. If you answer with a Y for Yes and follow with a (CR) or RETURN the system will ask for a Parameter Filename

EXAMPLE:

PARAMETER FILE MAKE? Y (CR)
PARAMETER FNAME...B:PFILE (CR)

A response of N or No and a (CR) or RETURN yields...

9. The system responds with

M E R G E S T A R T O K ?

If you answer Y or YES the MERGE process starts.

A typical result comment would be as follows:

INPUT-DATA 200 RECORDS
END OF MERGE RESULT IS B.MASTER.FL

A> (Back to CP/M prompt)

10. Note that input of parameter files is the same as for SORT parameter files described in the

previous section.

FDUMP Utility

Your SANYO computer has a utility which outputs the file contents either onto the CRT (display) or to the printer according to your keyed in parameters. There are two output forms to this utility:

1. Character Display
   The file contents are displayed with ASCII based characters. If the chosen character is not ASCII based a "?" will be displayed.

2. Hexadecimal Display
   The file is outputted in hexadecimal form.

The utility is invoked by the execution of FDUMP and (CR) or RETURN after the CP/M prompt message in the Operating System.

1. The call

```
A> FDUMP            (CR)

     FDUMP Ver 1.1
     FNAME,MODE,DEVICE,R-SIZE,END...
```

Here you must input

```
FNAME...Input file-name
MODE....A for characters
        H for hexadecimal
DEVICE..C for CRT (optional)
        P for printer output
        Pxx/yy
         xx= lines/page
             if omitted default
             value is 66.
             Otherwise 1 to 99.
         yy= columns/row
             if omitted default
             value is 80.
             Otherwise 1 - 255.
R-SIZE..Record length.
        Optional. Omited for
        sequential files.
END.....E to end dump at file's
        EOF flag byte (1A hex).
        If omitted output will
        continue to the final
        byte of sector after
        EOF flag seen.
```

EXAMPLE:

```
FNAME,MODE,DEVICE,R-SIZE,END...TEST.BAS,A,C
```

2. NOTES ON SYSTEM USE
Please observe the following:

a. After the dump process is over you may return to the operating system if you depress CTRL and C simultaneously

b. During the dump process CTRL + C will cause a pause in the dump process. To continue.... depress CTRL + C.

XREF Utility

Your SANYO computer has the capability to make a list of variables that are used in a program made by SANYO-BASIC and saved in the ASCII formats. In the listings, variables are shown as to their line attribute definition and how they are employed. The allowable number or variables is 240.

The XREF utility is called from the CP/M operating system after the proper prompt mesage has been displayed:

A> XREF (CR) or RETURN

1. The system responds with

```
XREF Ver 1.1
FNAME,MODE,DEVICE,DATE....
        Here you must input:
        FNAME...The input file-name.
        MODE....The output device form:
                A= Source and cross
                   reference form.
                S= Source File form
                X= Cross reference form
        DEVICE..Assigns the output to
                C= CRT Display tube
                P= Printer Output
                   Pnn can specify the
                   number of lines per
                   page (nn). The value
                   of 66 is default.

        DATE...This optional feature
               is keyed-in as
                        YY-MM-DD
                Where YY=year
                      MM=month
                      DD=date
```

2. After a (CR) or RETURN the system will start file listing. You may

a. Depress CTRL and C simultaneously to return to the operating system

mode.

b. Depress CTRL and S simultaneously to stop the listing. Pressing these two again will resume the listing.

XCHG Utility

Your SANYO computer has a very useful utility called XCHG. This allows one a function to replace characters and parameters in a program that has been made with SBASIC and saved in ASCII form. A new file is made with XCHanGed parameters.

The XCHG utility is called from the CP/M operating system after the proper prompt mesage has been displayed:

A> XCHG (CR) or RETURN

1. The system responds with
   XCHG Ver 1.1
   INPUT FNAME.....
   Here designate and input the filename to be operated on. Depress (CR) or RETURN after

2. The system responds with
   OUTPUT FNAME.....
   Here designate and input the new filename. If omitted the same name will be used. Depress (CR) or RETURN after

3. The system responds with
   CHANGE SYMBOL.....
   Here we change parameter names Let's use an example....

sss - eee /S/"xxx"="yyy", ..., ..., ...

Here we have
- sss .....Starting line number or if omitted the first line #.
- eee .....Ending line number or if omitted the final line #.
- S.....With S designated not only parameters but all equivalent characters are substituted. Without S designated only parameters are replaced.
- "xxx".....New parameter name.
- "yyy".....Old parameter name.
- ..., ...up to 5 parameter names can be separated with commas, each name can only be 20

columns wide.

EXAMPLE IN SBASIC FILE.....

```
A> XCHG
        XCHG Ver 1.1
        INPUT FNAME.......TEST.BAS
        OUTPUT FNAME......TEST.NEW
        CHANGE SYMBOL....."XX"="A", "YY"="B", "MM"="N"
```

This yields the following result:

TEST.BAS (Old file)

```
10 PRINT "TEST"
20 FOR N=1 TO 100
30 A=SQR(N)
40 B=A*N
50 PRINT N,A,B
60 NEXT
```

TEST.NEW (Changed file)

```
10 PRINT "TEST"
20 FOR MM=1 TO 100
30 XX=SQR(MM)
40 YY=XX*MM
50 PRINT MM,XX,YY
60 NEXT
```

4. If CTRL and C are pressed simultaneously as parameters are being keyed-in the system will respond by aborting the utility and goes back to the operating system mode.

COPYDISK Utility

It is always a good habit to copy master diskettes and diskettes with important data on them. This backup avoids problems that may arise if you mistakenly destroy programs or data on a diskette. The COPYDISK utility allows the copying of entire disks. Two drive must be present to make use of this utility.

Step 1 Insert the operating disk in drive A. Boot the system.

When the system input symbol (A>) is displayed, enter COPYDISK. The following prompt and instructions are displayed:

```
CP/M-86 Full Disk COPY Utility Version 2.0

Enter Source Disk Drive (A-D)?
Destination Disk Drive (A-D)?
```

Enter the source drive name (the name of the drive the disk is on that you want copied). The destination disk drive contains the blank disk.

The following prompt is displayed after you enter the drive names:

```
Copying Disk A: to Disk B:
Is this what you want to do (Y/N)?
```

You have the opportunity of exiting form the disk copy utility if you enter an N. If you wish to continue, enter Y .

The following prompts appear:

```
Copy started
Reading Track XXX
Writing Track XXX
Verifying Track XXX
```

These last three prompts will appear until the entire disk is copied.

You may copy another disk when the disk copying instructions are displayed again.

DDUMP Utility

This program outputs the contents of a disk on the CRT in units of tracks or sectors.

PROCEDURE

1) A > DDUMP CR

The following message will be displayed:

DDUMP - DISK DUMP - VER 1.0 DOUBLE

*

* indicates that DDUMP is in the command mode.

DDUMP - DISK DUMP

2) Key-in the following in the command mode (*):

*[drive name,]track number[,[sector number[,EDIT]] CR

Example: Dump the contents of sector 1 on tract 1 of drive B.

*B:,1,1 CR

	Drive name	Track No.	Sector No.
MBC-4000	A ~	0 ~ 76	1 ~ 24

3) By changing the mode to the edit mode, data can be change, 128 bytes at a time.

*1,1,EDIT

```
            DRIVE A - TRACK 1  SECTOR 1

0000 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0010 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0020 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0030 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0040 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0050 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0060 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0070 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
```

Example: Changing data at address 0, sector 1 on track 1

1) First, change the mode from the command mode to the edit mode.

2) When - is displayed, key-in the address where the data to be replace is written.

3) Replace data, one byte at a time.

4) To end replacing data, key-in a period.

```
-0              Specify an address (0 to 7F)
0000 E5 00      Replace (0 to FF)
0001 E5  .      End of replacement
```

```
            DRIVE A - TRACK 1    SECTOR 1

0000  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0010  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0020  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0030  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0040  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0050  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
```

```
0060  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
0070  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
```

5) To return from the edit mode to the command mode, key-in W or Q and press CR.

W ... Write replacement data on the disk and return to the command mode.

Q ... Return to the command mode without writing replacement data on the disk.

4) To return to the OS mode, press CTRL and C at the same time in the command mode.

ERROR DISPLAY

Disk read error	Data cannot be read from disk.
Disk write error	Data cannot be written on disk.
ADDRESS error	Invalid edit address
Error HEX input only	A non-hexadecimal character was input in the edit mode.
Invalid syntax Q (ABORT)	Extra data followed Q.
Invalid syntax W (WRITE)	Extra data followed W.

CHAPTER 14 - DIAGNOSTICS

Introduction

Your SANYO computer is a sophisticated result of man's mastery over electronics. There are ways to acertain that your computer is always in the prime of health. Likewise, you may experience problems at times that may suggest you take a closer look at the operational abilities of your computer. In this chapter we will focus our attention onto performing diagnostic and awareness procedures to help us with these tasks.

If we at times seem too simplistic in our approach please accept our apologies. At times the smallest of details, once overlooked, becomes the largest of problems. We will try to present a chronological sequence of questions for you to answer and follow with some diagnostic and trouble-shooting routines.

Get to know your computer as a close friend, take care of it and it will give you many years of performance and pleasure.

Trouble-Shooting List and Remedies

PROBLEM: COMPUTER SEEMS DEAD

1. Is the cooling fan on?
   If not there is no power to the unit;
   a. Is it plugged in?
   b. Power exists at the wall?
   c. Is the fuse blown?
   d. Is it turned on?

   A machine that turns on with no cooling fan action seriously needs attention. Cooling fans prolong the life of your computer.

2. Is the screen displaying anything?
   If not ask yourself
   a. Is the intensity turned down?
   b. Are we in a mode where there is all spaces on the display?
   c. Are we accessing files or is there any other thing happening?

PROBLEM: COMPUTER DISPLAY ERRATIC

1. Will the display clear up after a warm-boot?
2. Will the display clear up after a system RESET?

PROBLEM: COMPUTER DOESN'T DISPLAY WHAT IS KEYED IN

1. Is the KEYBOARD plugged into the computer and is it firmly attached?
2. Is the SHIFT LOCK activated?
3. Are you in GRAPHIC INPUT mode?

4. Is another key stuck down?

PROBLEM: COMPUTER INITIALLY TALKS TO DISKETTE BUT HANGS UP

1. Is the corect diskette mounted?
2. Is the diskette mounted correctly?
3. Is the door closed to the drive?
4. Does the program need to write to the disk and if so it the write protect on tab on the diskette?

PROBLEM: COMPUTER CANNOT COMMUNICATE WITH DISK DRIVE?

1. Is the correct diskette mounted?
2. Have you inputted the correct disk command?
3. Is the door closed to the drive?
4. Attempt a re-boot or RESET operation to re-try.

PROBLEM: ERRATIC OPERATION OF DISK DRIVE...AT TIMES

1. Are you using the correct type of diskette?
2. Are your diskettes dirty or worn?
3. What is the storage temperature of system?
4. Are your diskettes subject to smoke films?
5. Are the diskettes bent or warped?
6. Are the heads in the drive dirty or need adjustment?
7. Are the diskettes part of your system or from another?

PROBLEM: PRINTER PROBLEMS

1. Is the printer cable firmly attached?
2. Is there a fault in the printer? (paper out, ribbon)
3. Do you have a new ribbon?
4. Is the paper set correctly?
5. If your form is software supported...is it correct?
6. Is your printer on and enabled?

CHAPTER 15 - INPUT / OUTPUT SYSTEMS AND CONNECTORS

Introduction

Your SANYO sytem has a plurality of connectors that make it a very universal computer in every sense. This chapter will cover the input and output hardware parameters in more detail.

The Printer Connector

All MBC series computers feature a parallel printer connector. This connector allows the user to attach a high-speed printer peripheral to their SANYO. Printers require data that is ASCII in nature. This means our connection scheme should have 8 lines or bits of available information. The printer port exists on the back of your machine and is implemented in an industry-standard 36 pin connector. The actual pinouts are shown below:

PIN NO	SIGNAL NAME	DESCRIPTION
1	LPSTB	Strobe level low to printer. Means data on line is valid.
2	DATA1	Data line to Printer
3	DATA2	Data line to Printer
4	DATA3	Data line to Printer
5	DATA4	Data line to Printer
6	DATA5	Data line to Printer
7	DATA6	Data line to Printer
8	DATA7	Data line to Printer
9	DATA8	Data line to Printer
10	ACKNOWLEDGE	Handshake Line from printer to CPU
11	BUSY	Handshake Busy from printer to CPU
12-18	No Connection	
19-29	Logic Ground	Ground Reference
30-36	No Connection	

In operation the computer will place data onto the 8 DATA lines and then will assert the term LPSTB, a low going term. This tells the printer that the computer is ready to output the data through the port. An ACKNOWLEDGE and BUSY line from the printer tell the computer that the information has been received and that the print operation is in process.

The Serial Port

All MBC computers have serial communication ability. In the MBC-4000 series there is a single RS-232-C port.

The serial protocol is standard RS-232-C as per the industrial standard. The way information is exchanged (as well as pin number designations) is defined in detail in the technical reference section.

Locations of the various connectors are detailed in the technical reference section.The connectors are called a "D" type due to the shape. They are a DB25 Female, which is the standard for the CPU end. The voltage levels are +12 and -12.

The Keyboard

The KEYBOARD attaches to the rear of the unit with a special connector that must be firmly attached. Insure that the two holding screws are snugly screwed in to establish the connection.

The KEYBOARD is scanned by the computer in a row and column format. This means several parallel scan lines are activated through the interface and the resultant parallel code that results from the scan is returned back to the processor.

It is important to know how the KEYBOARD differs peripherally. The KEYBOARD is scanned only when the processor decides to do so. The KEYBOARD is a static, passive device that has not holding or memory buffer locally. If the processor is not looking at the KEYBOARD PORT no awareness will be had as to key depression. This means the KEYBOARD may be "masked out" by software directives. Further, there is no magic reason any key depressed should equal that configuration on the screen. This means that other characters may at times appear on the screen for the key that is depressed, depending on the software protocol. This may be normal procedure for some softwares.

The programmable function keys are user defined in BASIC only. We recommend your labeling the keys for specific programs. Details for these keys appear in Chapter 9.

Please keep in mind the one-at-a-time nature of the KEYBOARD. Two keys pressed simultaneously (to you) will be scanned in sequence by the computer. Fast key input may be subject to errors in translation. The computer may effectively process about 20 characters per second.

Configuration Control of the RS-232C Port

The serial interface in the MBC-4000 series is a small logic module located within the computer. It is placed into a logic card cage. You have access to it by removing the top of the computer. We will show you how to change the baud rate and address of this interface.

CHAPTER 16 - INPUT / OUTPUT SOFTWARE AWARENESS

Introduction

The various I/O structures are accessible under software control both from the operating CP/M system as well as from the SBASIC that comes with every MBC computer. Here we will focus on the aspects of dealing with the software delegations and commands for the I/O system in general. This chapter is primarily an index to help point you to other sections of this manual.

Peripheral Use

PRINTER

The printer is accessed from the PIP command directly from CP/M or through other PRINT handlers.
See Chapter 18.
The printer is accessed from SANYO-BASIC through commands prefixed with an "L"....example LLIST and LPRINT.

SERIAL PORTS

Under CP/M these ports can be PIPed or used with other commands. Use STAT to re-delegate CP/M calls.
See Chapter 6.
Under SANYO-BASIC the serial ports work with commands that contain "%" in their suffix.....examples:

INIT%, INPUT%, LINEINPUT%, PRINT%

See Chapter 9 and the technical reference section (Input/Output firmwares for a better understanding of serial ports.

KEYBOARD CONTROLS

See the technical reference section, Input/Output Firmwares and Chapter 15. The KEYBOARD is not generally available as a user defined peripheral except in the normal input sense.

CHAPTER 17 - SPECIFICATIONS

Hardware

CPU

8086 (5MHz, maxim mode)

NDP

8087 (optional)

PIC

8259A

MEMORY

RAM	128KBytes (standard), 384KB (option)
ROM	12KB

KEYBOARD

Structure	Separate type, step key is used, serial ASCII specifications
EXTERNAL STORAGE	Mini-floppy disk, double sided, double density, double track, soft sector.
	Capacity: 640 KBytes (when formatted)
	Optional mini-floppy disk (640 KBytes)
Power Source	Local voltage $\pm$ 10%
External Dimensions	Mainframe: 442(W) x 328(H) x 347(D)mm
	Keyboard : 442(W) x 63(H) x 246(D)mm

Display

Display Method	Raster scanning method
CRT	12 inches, green, non-glare screen
Number of Displayable Characters	8 characters x 25 lines
Character Configuration	8 x 12 dot matrix

Display Letter Type	256 types
Cursor	Block reverse blink
Letter Modification Type (attribute)	Reverse blink, background Inverse, blink, background

Interface

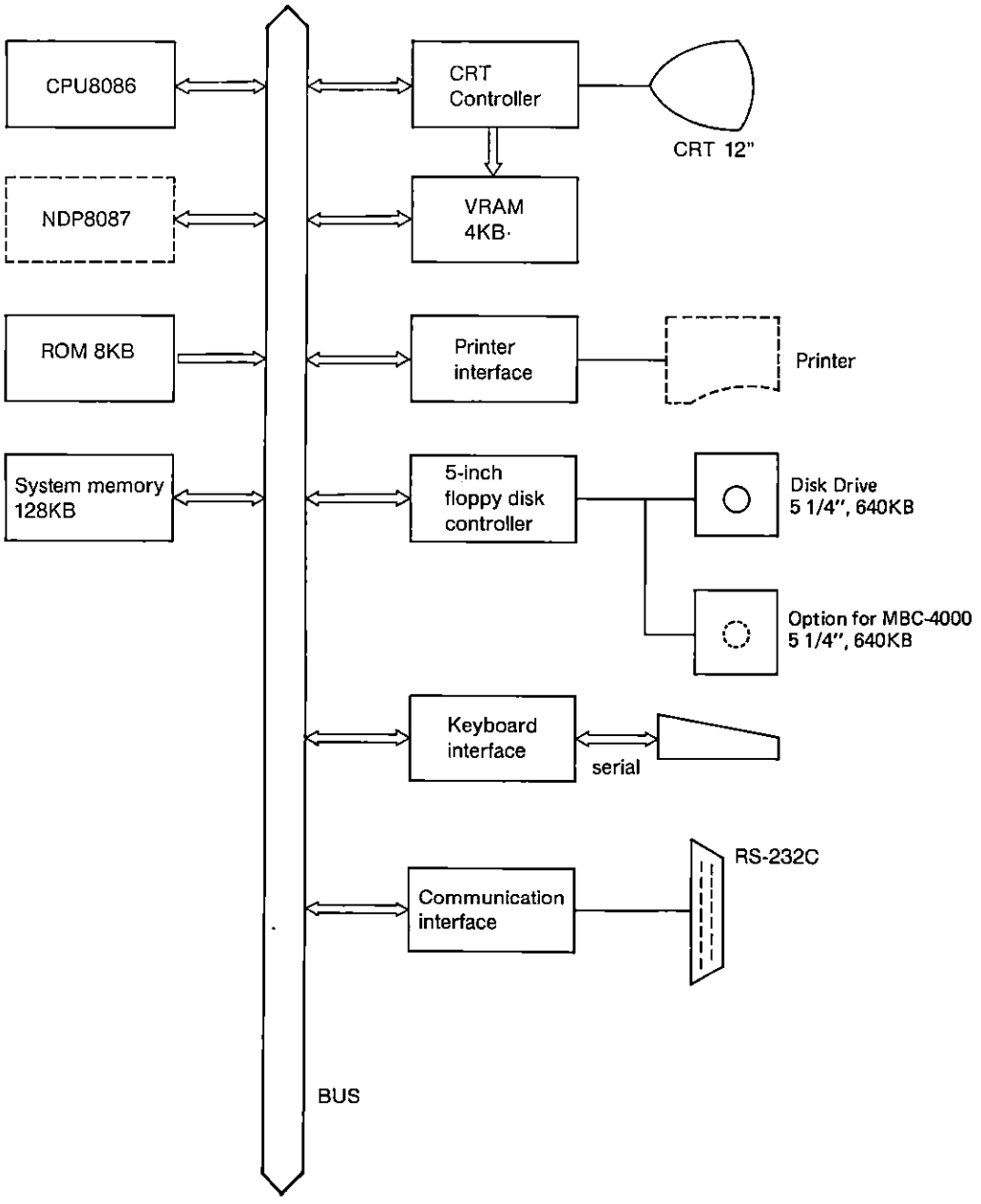
Serial Interface	RS232 port (1)
Mode	Asynchronous
Speed	1200 - 4800 bauds
Printer Interface	Parallel I/F (Centronics)

Software

Operating System	CP/M-86 V.1.1
Language Processor	SANYO BASIC Other languages permitted are PASCAL/M, COBOL-80 FORTRAN-80, MACRO-80, BASIC-80 and application software run under CP/M.

HARDWARE CONFIGURATION

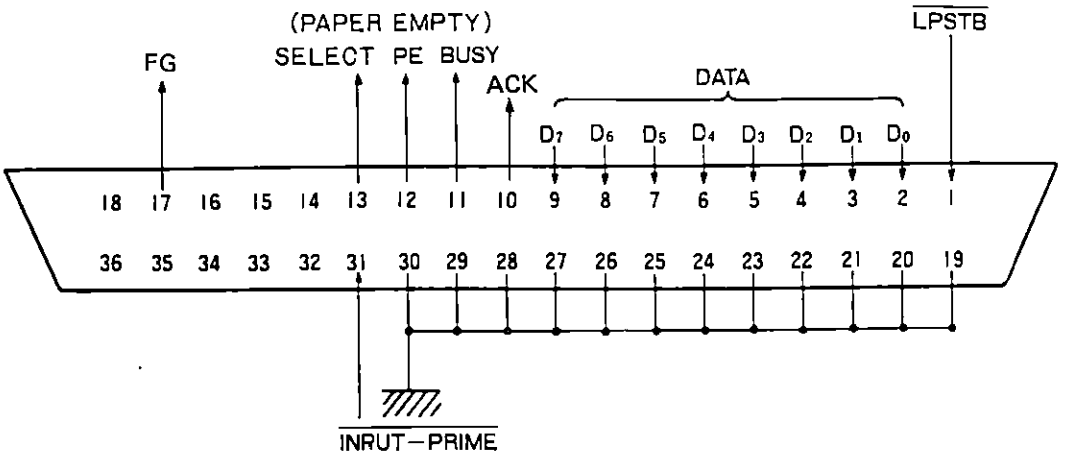
(DIAGRAM)



Connector Pins

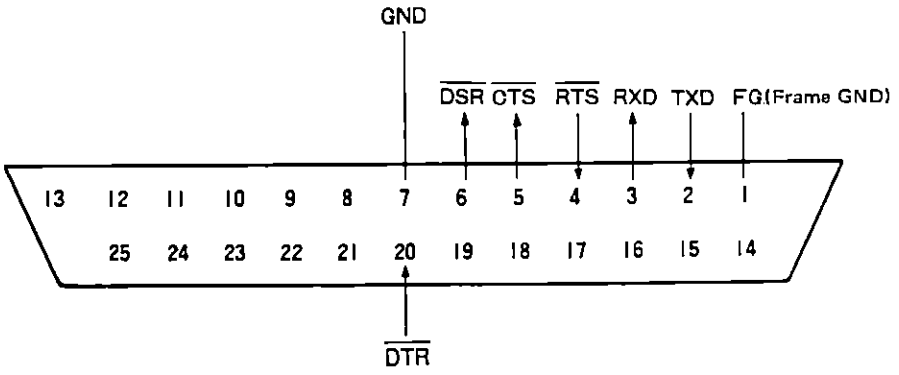
Connector pins for serial and parallel ports

Printer (parallel I/F)



In 7417, the printer output is pulled up. A combined 3.3KΩ resistance is used.

Serial (RS-232C)

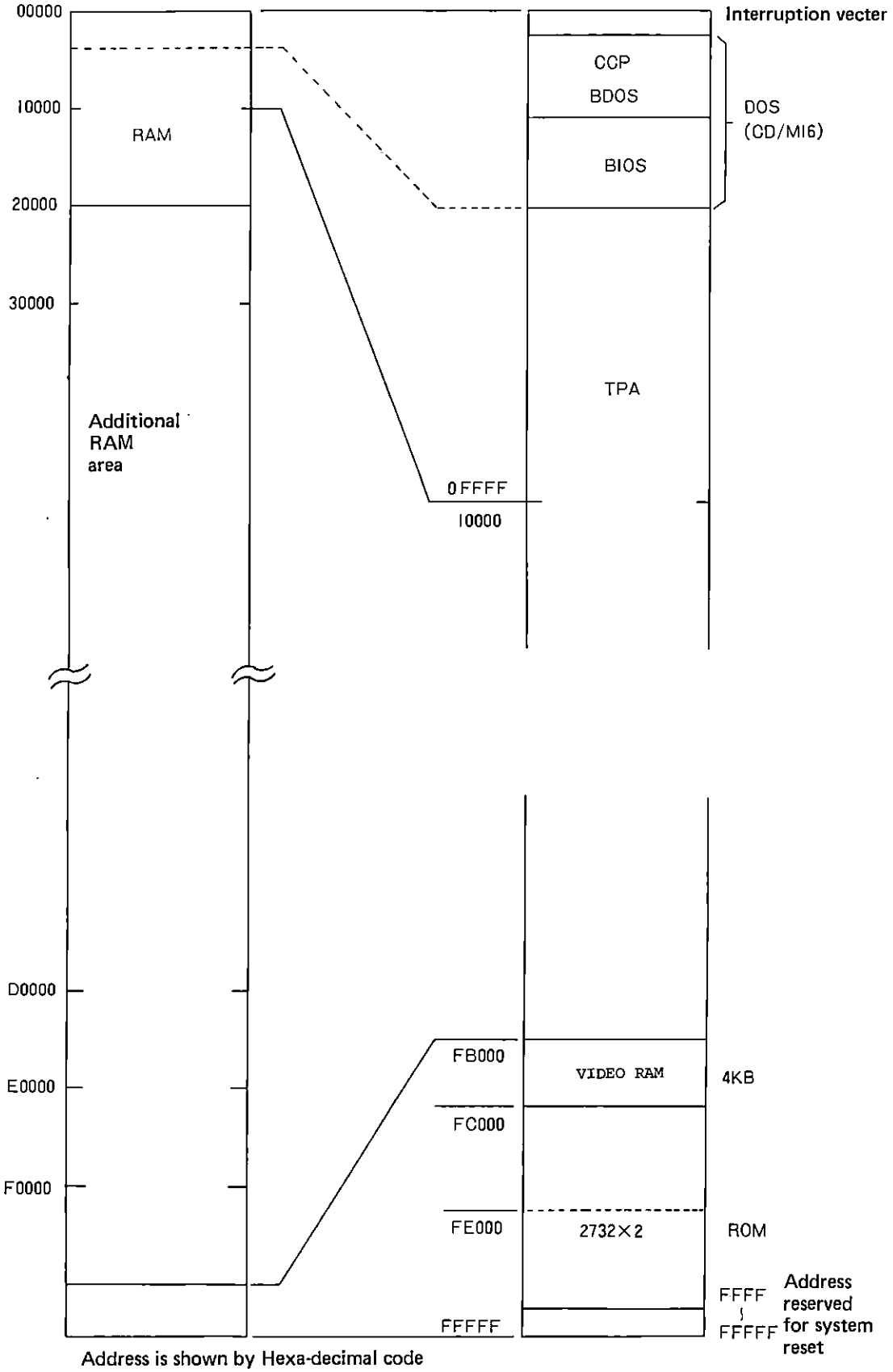


Use the following connectors:

For printer : 57-30360 (Daiichi Denki Kogyo) or an equivalent product

For RS232C: DB-25P (Nippon Koku Denshi) or an equivalent product

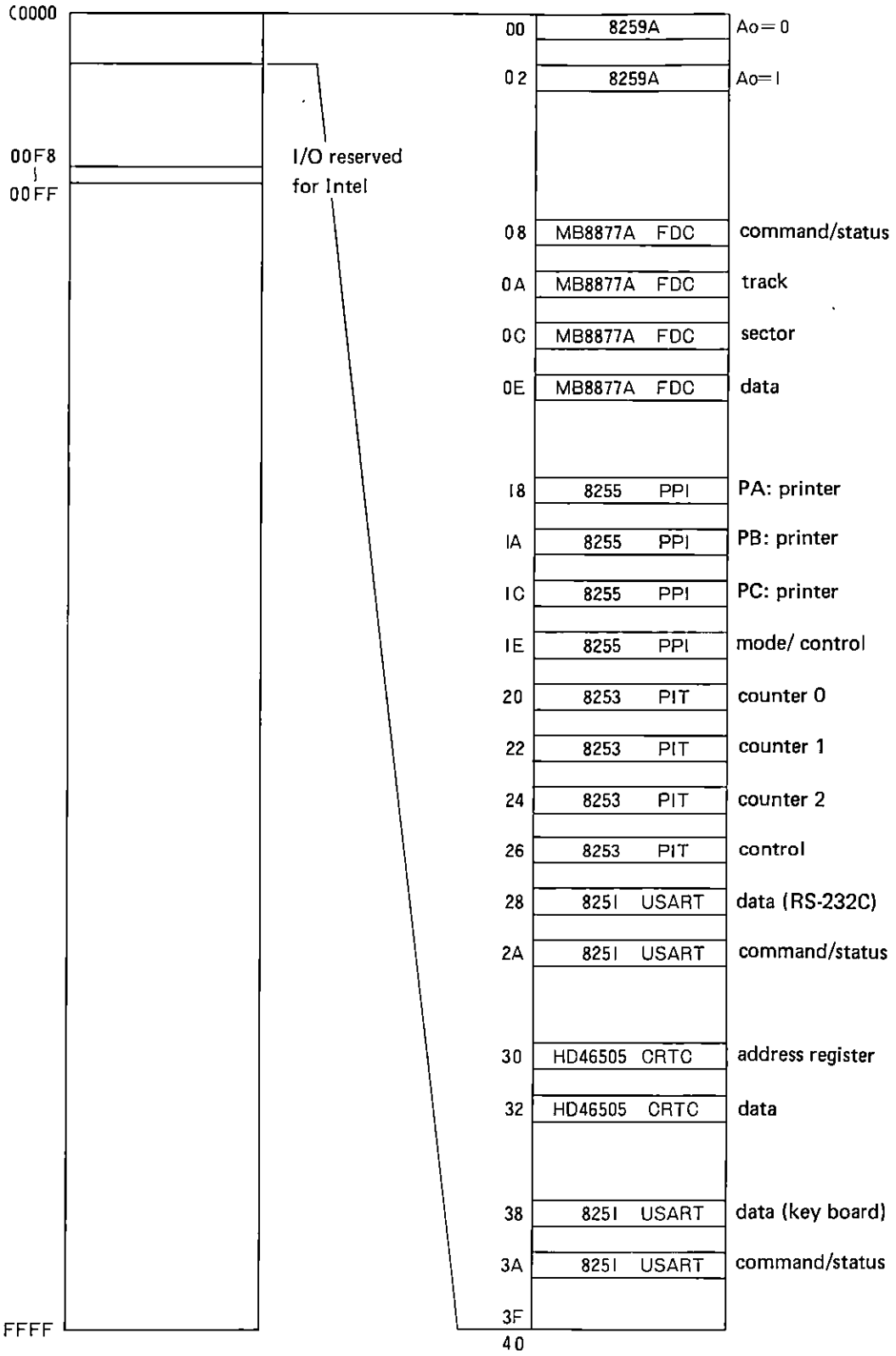
Memory Map for Main CPU



Address is shown by Hexa-decimal code

I/O Map

I/O address.



BIOS Interface

Jump vector of BIOS

Offest from the beginning of BIOS	Jump vector
2500H	JMP INIT
2503H	JMP WBOOT
2506H	JMP CONST
2509H	JMP CONIN
250CH	JMP CONOUT
250FH	JMP LIST
2512H	JMP PUNCH
2515H	JMP READER
2518H	JMP HOME
251BH	JMP SELDSK
251EH	JMP SETTRK
2521H	JMP SETSEC
2524H	JMP SETDAM
2527H	JMP READ
252AH	JMP WRITE
252DH	JMP LISTST
2530H	JMP SECTRAN
2533H	JMP SETDMAB
2536H	JMP GESTEBG
2539H	JMP GETIOB
253CH	JMP SETIOB

RS-232C serial interface

1. 8251A is used for USART.
2. Address for 8251A is as follows:

Data port	28H
Command status port	2AH

3. TxRDY and RxRDY for 8251A are connected with 8259A interruption controller.

TxRDY	8259A IR1	249 (F9H)
DxRDY	8259A IR2	250 (FAH)

4. Transfer clocks are generated by the counter 2 in 8253.
   Baud rate can be changed by programming of 8253 counter.
5. Initial setting is as follows:

start bit	1 bit
data bit	8 bits
parity	disable
stop bit	1 bit
baud rate	1200 bauds
asynchronous method	

Setting of Baud Rate

It can be changed by writing new setting value into the counter 2 of timer/counter.

ASSEMBLER

```
BPS     EQU     4800        ; BAUD RATE
RSBR    EQU     24H         ; COUNTER 2
RATE    EQU     12500/ (BPS/100)
;
        MOV     AX, RATE
        OUT     RSBR , AL   ; LSB
        MOV     AL, AH
        OUT     RSBR , AL   ; MSB
```

BASIC

```
10      BPS=4800     :REM      BAUD RATE
20      RSBR=ZH24    :REM      COUNTER 2
30      RATE=12500/(BPS/100)
40      AL=RATE AND 255
50      AH=(RATE/256) AND 255
60      OUT RSBR, AL :REM LSB
70      AL=AH
80      OUT RSBR, AL :REM MSB
```

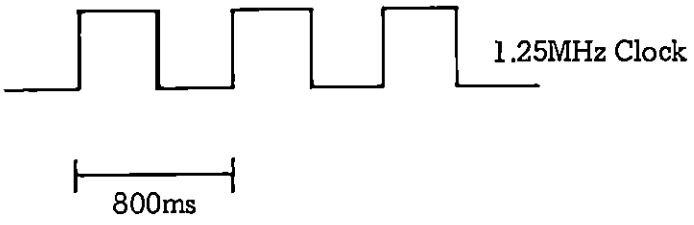
Timer

1. 8253 is used for counter/timer.

2. Address of 8253 is as follows:

Counter 0	20H Connect with 8259A (Interruption controller) IRO. Initial setting value is 10 msec period.
Counter 1	22H Connect with speaker
Counter 2	24H RS-232C baud rate setting. Initial value is 1200 bauds.
Control port	26H

3. Clock frequency for 8253 counter is 1.25 MHz.



Interruption controller

1. 8259A is used for the interruption controller.

2. 8259A has 8 levels of interruption request terminal, and interruption request from each terminal can be detected at the edge. Assignment for interruption request terminal is as follows:

Terminal	Vector	
IR0	248 (F8H)	interval timer (settled to 10 msec)
IR1	249 (F9H)	USART TxRDY (RS-232C)
IR2	250 (FAH)	USART RxRDY (RS-232C)
IR3	251 (FBH)	USART RxRDY (Keyboard)
IR4	252 (FCH)	printer READY
IR5	253 (FDH)	floppy disk controller
IR6	254 (FEH)	8087 numerical data processor (option)
IR7	255 (FFH)	user interruption (negative logic)

3. Priority order for each terminal is fixed in the order from IR0 (high) to IR7 (low). Priority order can be changed by OCW (Operation Command Ward) 2.

4. Only IR2 and IR3 correspond to BIOS. Other terminals than IR2 and IR3 among IR0 to IR7 are masked with 8259A internal interruption mask register (IMR). IMR can read out and write in with READ/WRITE corresponding to I/O address 02.

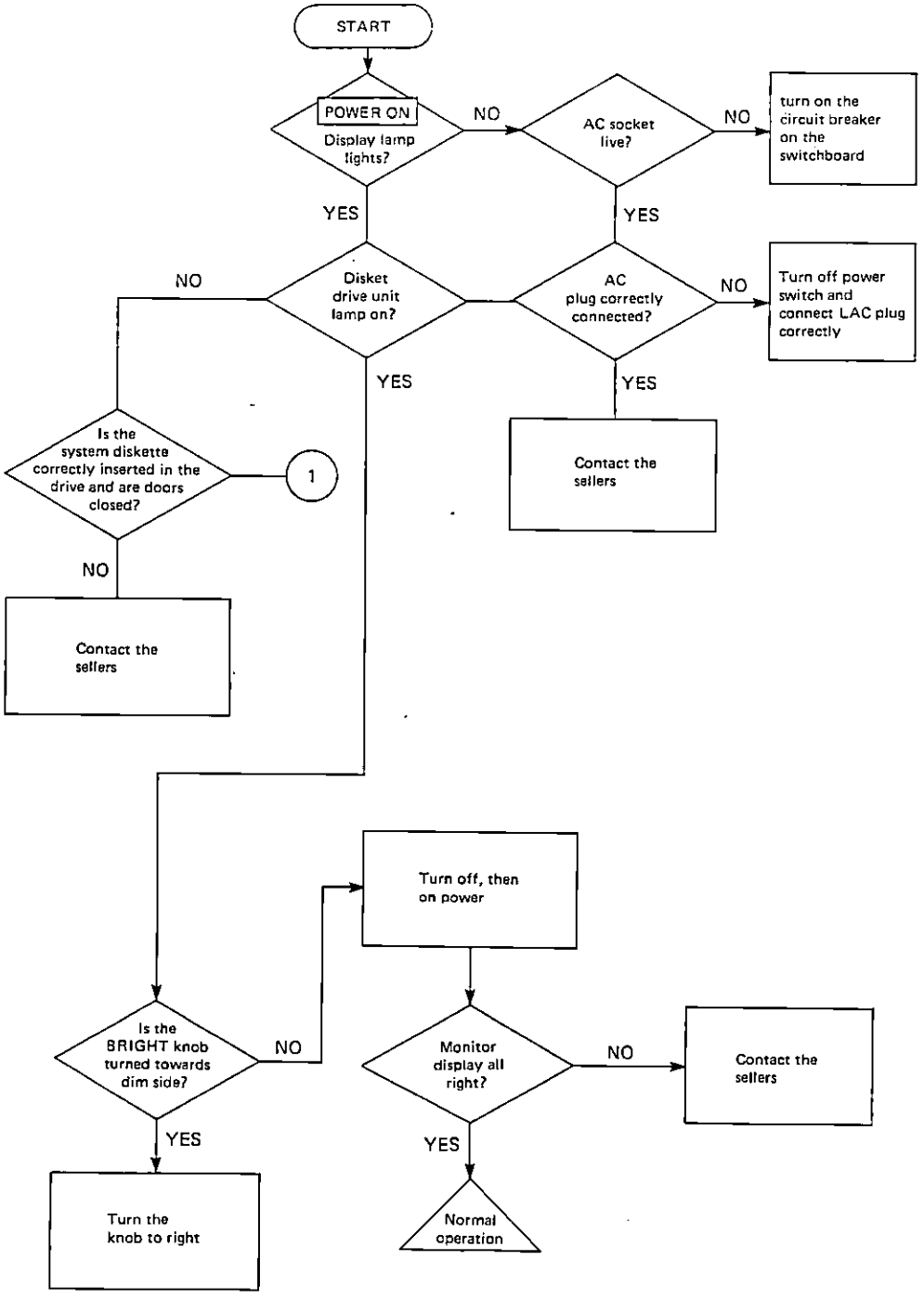
   (Note) When masking IR2 or IR3, response may be slow because of software polling. In the inside of BIOS, the interruption is inhibited only for the input/output routing with floppy disk. During the execution of this routine, an interruption can not be accepted. (about 230 msec for maximum)

5. Interruption rather than IR2, IR3 and IR5 among IR0 to IR7 can be used by transient program. In order to use the interruption, write segment and offset for interruption service routine into the corresponded vector, and reset the corresponded bit of IMR. When the program execution has been completed, put back the rewrited vector to the original location, and set the bit of IMR which has been reset, and then return the control right to DOS.
If vector or IMR is not recovered, it may cause uncontrollable running.

(Note) Please use the interruption upon your fully understanding of the operation for 8086 and 8259A.

Simple Trouble-shooting

If a trouble occurs, proceed as shown in the flowchart. For the error messages, see the respective manuals. No monitor data is displayed on the screen (during cold start)



TECHNICAL REFERENCE

LIST OF RESERVED WORDS

The following are reserved words which have some meaning as functions or statements. These words should not be used as variables in your programs or as labels in any form. In fact, your variables cannot have these words as pieces of your text.

ABS
ALL
AND
AS
ASC
ATN
ATTR$
AUTO

BASE
BEEP

CALL
CDBL
CHAIN
CHR$
CINT
CLEAR
CLOSE
CLS
COMMON
CONT
COS
CREATE
CSNG
CSRLIN
CVD
CVI
CVS

DATA

DEF
DEFDBL
DEFINT
DEFSEG
DEFSNG
DEFSTR
DELETE
DIM
DSKF

EDIT
ELSE
ERASE
ERL
ERR

ERROR

EXP

FIELD
FILES
FIX
FLD$
FN
FOR
FRE

GET
GO
GOSUB
GOTO
HCOPY
HEX$

IF
IMP

INIT
INKEY$
INP
INPUT
INPUT$
INSTR
INT

KEY
KILL

LEFT$
LEN
LET
LFILES
LINE
LIST

LLIST
LOF
LOG
LPOS

LPRINT

LSET

MERGE
MID$
MKD$
MKI$
MKS$
MOD

NAME
NEW
NEXT
NOT

OCT$
ON
OPEN
OPTION

OR
OUT

PACK$
PEEK
POKE

PRINT
PUT
POS
RANDOMIZE

READ
REM
RENUM
RESET
RESTORE
RESUME

RETURN
RUN

SAVE
SEG
SET
SETKEY
SGN
SIN
SPACE$
SPC
SQR
STEP
STOP
STR$
STRING$
SUB
SWAP
SYSTEM

TAB
TAN
THEN

TINPUT
TO

TROFF
TRON

UNPACK$
USING
USR

VAL
VARPTR

WAIT

WEND
WHILE
WIDTH
WRITE

XOR

END	LOAD	RIGHT$
EOF	LOC	RND
EQV	LOCATE	RSET

CHARACTER CODES FOR THE SANYO

Explanation

The chart below depicts the character codes for the SANYO computer series. You will note immediately there are 256 possible codes. The character set is a combination of both the standard 7-bit ASCII set and the graphic characters that are on the second half of the set (with the 8 bit at a high state). The configuration for the digits in the X dimension repesent bits:

B7 B6 B5 B4

and the bits in the Y (vertical) dimension represent the bits:

B3 B2 B1 B0

The combination of these two "nybbles" represent the character code byte.

				B7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				B6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				B5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				B4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
B3	B2	B1	B0		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NU	DE		Ø	@	P	`	p			∂	—	à	∫		
0	0	0	1	1	SH	D1	!	1	A	Q	a	q			∘	¹	α	θ		
0	0	1	0	2	SX	D2	"	2	B	R	b	r			⌐	û	β	0		
0	0	1	1	3	EX	D3	#	3	C	S	c	s			£	ü	γ	√		
0	1	0	0	4	ET	D4	$	4	D	T	d	t			â	ù	δ	‰	♠	⊥
0	1	0	1	5	EQ	NK	%	5	E	U	e	u			ä	Ä	λ	∞	♥	⊤
0	1	1	0	6	AK	SY	&	6	F	V	f	v			á	É	ε	ζ	♦	⊣
0	1	1	1	7	BL	EB	'	7	G	W	g	w			ã	Ö	ω	ψ	♣	⊢
1	0	0	0	8	BS	CN	(	8	H	X	h	x			ê	Ü	Σ	φ	—	┼
1	0	0	1	9	HT	EM	)	9	I	Y	i	y			ë	Ã	Ω	∀	|	★
1	0	1	0	A	LF	SB	*	:	J	Z	j	z			î	ñ	Δ	∃	═	▼
1	0	1	1	B	VT	EC	+	;	K	[	k	{			ï	Ñ	π	∘	╳	±
1	1	0	0	C	FF	←	,	<	L	\	l	:			ì	Å	π	ç	═	≥
1	1	0	1	D	CR	→	-	=	M	]	m	}			ô	€	μ	§	═	≤
1	1	1	0	E	SO	↑	.	>	N	^	n	~			ö	≈	ρ	¨	/	÷
1	1	1	1	F	SI	↓	/	?	O	_	o				ò	≠	ζ	°	/	▲

The Character Code Set

ERROR MESSAGES

Errors are checked at the time of editing (Editing Error). When the program is generated and keyed in, a simple check is performed to determine whether or not the program can be loaded into memory. All the syntactical checks in detail are performed at the time of execution.

The following items are tested at the time of editing:

- o Is the string data used properly?
- o Do the number of lines exceed 65535?
- o Can the keyed-in program be loaded into memory?

Errors at the time of execution (Execution Error)

Errors other than the above are all checked at the time of execution. When an error is found, the system displays the corresponding error message on the screen and returns the control to the BASIC Mode. When the error trap routine exists in the program, the control jumps to the error handling subroutine.

Error messages to be displayed during editing or execution are shown below.

BASIC Error Message List

NO.	CODE	MESSAGE	EXPLANATION
1	01	NEXT without FOR	FOR was not found in the FOR/NEXT loop.
2	02	Syntax error	Syntax error.
3	03	RETURN without GOSUB	RETURN was found although the GOSUB was not executed.
4	04	Out of DATA	Number of constants in the DATA statement is less than the number of variables in the READ statement.
5	05	Illegal function call	Illegal statement or function call was found.
6	06	Overflow	Numeric value overflow.

7	07	Out of memory	Memory was exhausted. Also too large an array
8	08	Undefined line number	Specified line number was not found.
9	09	Subscript out of range.	Array subscript was out of the range.
10	0A	Duplicate definition	The definition was duplicated. Two DIMENSION statements for the same variable.
11	0B	Division by zero	Number divided by 0.
12	0C	Illegal direct	It cannot be directly executed A statement illegal in the direct mode.
13	0D	Type mismatch	Variable type does not match the data type.
14	0E	Out of string space	The character string area was exhausted.
15	0F	String too long	The character length exceeded 255 characters.
16	10	Printer not ready	The printer is not ready to
17	11	Can't continue	Cannot be continued (CONT).
18	12	Undefined user function	The user function was not defined.
19	13	No RESUME	RESUME was not found in the error processing routine.
20	14	RESUME without error	RESUME was executed although no error has occurred.
21	15	Unprintable error	Undefined error.
23	17	Line buffer overflow	The entered line was too long.
24	18	Position not on screen	The position is out of the screen.
25	19	Print field overflow	The data cannot be stored in the specified print field.
26	1A	FOR without NEXT	NEXT was not found in the FOR/NEXT loop.

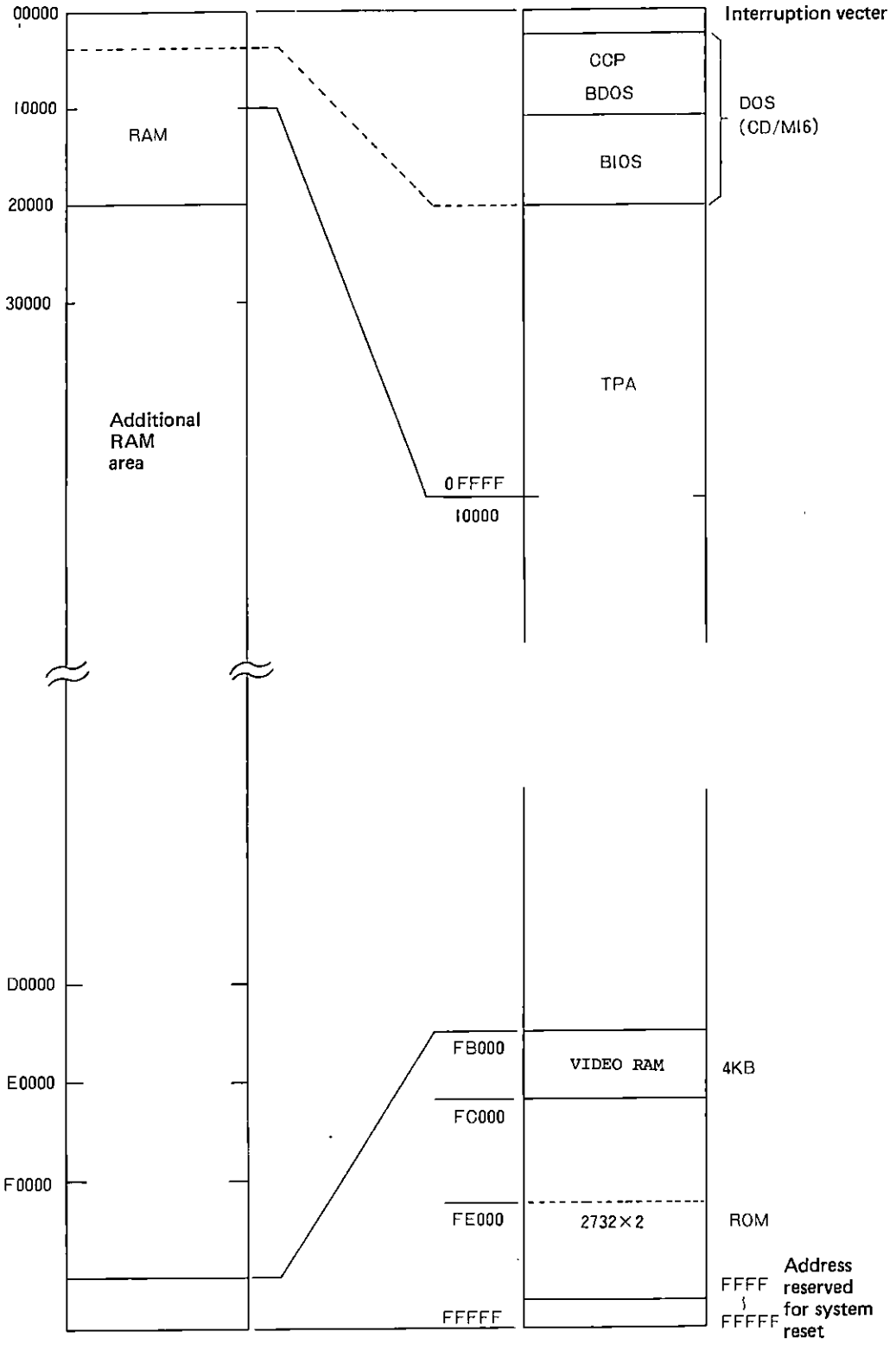
29	1D	WHILE without WEND	WEND was not found in the WHILE/WEND loop.
30	1E	WEND without WHILE	WHILE was not found in the WHILE/WEND loop.

DISK ERROR MESSAGES

50	32	Field overflow	The FIELD length exceeded the specified record length. More than 255 characters.
52	34	Bad file number	The file number is invalid.
53	35	File not found	The specified file was not found.
54	36	Bad file mode	The file open mode was not found.
55	37	File already open	The file was already opened.
56	38	Bad file data	The file data is invalid.
57	39	Disk I/O error	The disk is invalid.
58	3A	File already exists	The specified file already exists.
59	3B	Disk not ready	The disk device is not ready to operate.
60	3C	Bad disk select	Nonexistent drive specified.
61	3D	Disk full	The disk is full; no space to write.
62	3E	Input past end	The write instruction was found after the end of file.
63	3F	Bad record number	The record number is invalid.
64	40	Bad file name	The file name is invalid.
67	43	Too many files	Too many files are opened.
68	44	File write protected	The file is write protected or the diskette has been replaced.
69	45	NAME across disk	The NAME was incorrectly specified.

MEMORY MAPS

It is the intent of this section to present the memory map structure for the SANYO MBC 4000 series.



Address is shown by Hexa-decimal code

INPUT/OUTPUT FIRMWARES

Introduction

It is at times easier to talk to your SANYO series computer in it's native language.....the language of the machine code program. To do commands must be executed either from SANYO-BASIC or CP/M that can access the input and output structures of your machine directly. In this way extremely fast and effective programs may be written.

Many of you have awareness of system structures at the machine code level. This appendix is for you. We will present the input/output (here-after called the I/O) system to you in it's raw address form.

The subject matter here is a of a technical nature. It is not the intent of this appendix to elaborate on any specify hardware structure but to present an overview of the general architectures.

MBC-4000 Series I/O Structure

The MBC-4000 series is based on a 8086 based micro-processor and as such fully supports the I/O architecture of such a system. In that we mean the system can be mapped to support 65536 I/O addresses, from 0000(Hex) to FFFF(Hex).

1) Keyboard Interface

Data Port	38H
Control Port	3AH

2) RS232C Serial Interface

Data Port	28H
Control Port	2AH

3) Parallel Interface

For PORT A of 8255	18H
For PORT B of 8255	1AH
For PORT C of 8255	1CH
For Mode of 8255	1EH

PORT A is the output port for the parallel PRINTER PORT.
PORT B is the parallel control output port for the FLOPPY DISK.
PORT C is the port configured for the PRINTER CONTROL PORT.

4) Floppy Disk Controller the bits mean:

For Read/Write Status	08H
For Read/Write Track	0AH
For Read/Write Sector	0CH
For Read/Write Data	0EH

5) Timer Counter(8253)

Counter	0	20H
	1	22H
	2	24H
Control		26H

6) CRT Controller

Address Register	30H
Data Register	32H

7) INT Controller(8259)

	00H
	02H

MBC-4000 SERIES USE OF RS-232-C PORTS

The RS-232-C interface of the SANYO MBC-4000 series is implemented onto a separate printed circuit board that plugs into a "mother-bus" structure within the machine. This board has several options which will be covered in this section. For a detailed reference to the workings of RS-232-C, refer to the following section, THE RS-232-C INPUT/OUTPUT PORTS.

Most all RS-232-C interfaces are implemented with a family of integrated circuits refered to as "8251" types. These devices are universal asynchronous receiver/ transmitters often called UARTS. The UART in the architecture is a 8251 type.

UARTs are condusive to serial communication. They contain a serial-to-parallel converter and vice versa. They also have status and control registers that allow one to monitor data flow. Typically all UARTs have 4 registers:

1. DATA OUT Buffer, an eight bit out-data latch.
2. DATA IN Buffer, an eight bit in-data latch.
3. CONTROL Register for UART system allocations.
4. STATUS Register for line awarenesses.

In use one latches data into the DATA OUT Buffer from the system bus. Data is then serialized out of the I.C. to the external world. Meanwhile, serial data flows from the external world into the UART in serial. When an eight bit word has been received the 8251 latches the Data and signals the processor a READY condition. Data is then available in parallel at the DATA IN Buffer. During this time the UART is also responsible for data and control awareness at the port level to other devices. It does this through various "handshake" lines.

Upon initialization the proper mode must be sent to the 8251 UART to allow it to understand how to configure itself and future communications. This process of control signals define the complete functional definition of the 8251 and must follow immediately after a RESET operation

Software drivers (handlers) are in your SANYO system to accomplish this auto-matically upon system power up. These are in the CP/M BIOS (or Input/Output) support system.

There are four groups of I/O addresses allocated in the SANYO system for the RS-232-C ports. Each group is for RS-232-C ports that have been strapped to accept that particular address group and no other. The four groups are;

GROUP (JP #)	8251 MODE,COMMAND,STATUS	8251 DATA
1 ON	A5 (HEX)	A4 (HEX)
2 ON	AD (HEX)	AC (HEX)
3 ON	B5 (HEX)	B4 (HEX)

4 ON	BD (HEX)	BC (HEX)

NOTE: Only ONE JP # ON at a time.....all others OFF!

Most all systems are shipped with A4 and A5 selected.
This means to lock at data do an I/O read at A4 (Hex). To output data do an OUT command (I/O write) to port A4 (Hex). Reading port A5 gives one the STATUS information byte. Writing A5 delegates the MODE and COMMAND Parameters.

Use the INP command in decimal to read the port.
Format for the command is;

Variable = INP(xxx)

The following table should help you in asserting the meaning of each bit configuration.....please refer to technical manuals published by the various integrated circuit suppliers for complete details as to the use of the 8251 family of devices:

READ BIT MODES OF 8251	STATUS	DATA
B0	Transmit Ready	DO
B1	Read Ready	D1
B2	Transmit Buffer Empty	D2
B3	Parity Error Flag	D3
B4	Over-run Data Flag	D4
B5	Framing (Timing) Err	D5
B6	SyncDetect	D6
B7	Data Set Ready Flag	D7

WRITE BIT MODES	COMMAND	MODE
B0	Transmit Enable	Baud B1
B1	Data Terminal Rdy Out	Baud B2
B2	Receive Enable	Length L1
B3	Send Break Character	Length L2
B4	Error Reset (OE,FE,PE)	Parity Enable
B5	Request to Send Out	Even Parity On
B6	Internal Reset	Stop Bits S1
B7	Enter Hunt Mode	Stop Bits S2

For The B1 and B2 Baud determinants:

B1	B2	
0	0	Synchronous Mode
1	0	X1 Baud Rate Divider
0	1	X16 Baud Rate Divider
1	1	X64 Baud Rate Divider

For the LI and L2 Length Bits

0	0	5 Bits
1	0	6 Bits
0	1	7 Bits
1	1	8 Bits

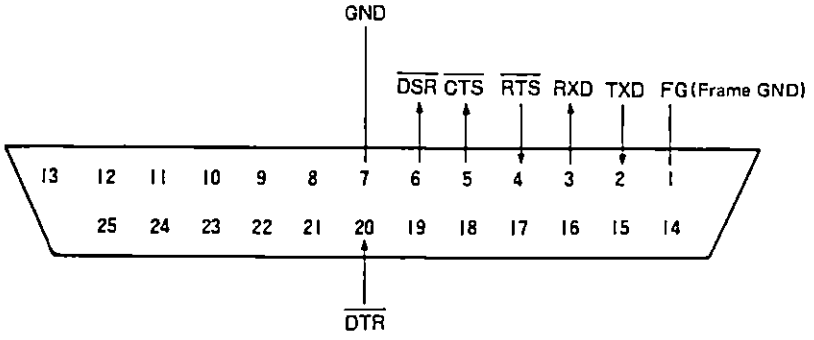
For the S1 and S2 Stop Bits

0	0	Invalid
1	0	1 Bit
0	1	1 1/2 Bits
1	1	2 Bits

THE RS-232C INPUT/OUTPUT SERIAL PORT

The Sanyo 4000 series communicates with electronic peripherals through special ports called RS-232C. This is an interface that involves very few conductors and allows two or more devices to communicate along a serial path.

The 4000 series has one RS-232 port.



Many devices may share the same serial line, a "star" or a cluster arrangement may be formed. Usually RS-232-C devices are printers or telephone acoustic data couplers called modems.

RS-232-C sends information on primarily one wire to the device and listens for a response on another wire. Other signals (or wires) are used for ready/busy awareness.

The following review will define RS-232-C. Although somewhat technical, this section is designed as a reference guide for overall use.

A Review of What RS-232-C Is

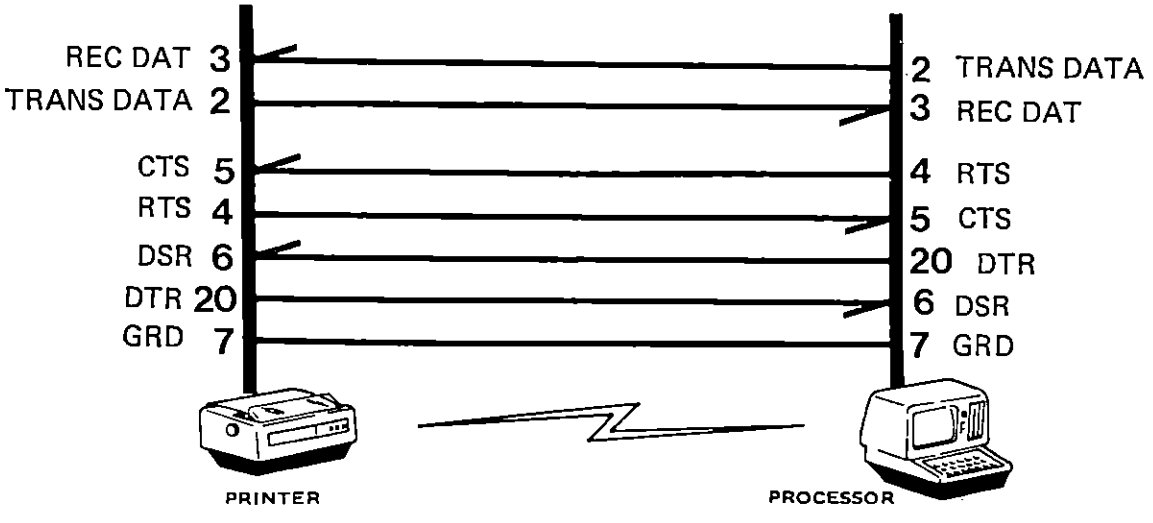
RS-232-C is a technical standard of the Electronic Industries Association (EIA). Published in August of 1969, it is titled "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange." The standard specifies electrical signal characteristics and names and defines the functions of the signal and control lines which make up a standard interface called RS-232-C.

The RS-232-C standard defines some 20 signaling lines or "circuits", as the standard refers to them. Most of them are optionally used. Even with many omissions and deviations from the standard, a link may still be referred to as "RS-232-C". It is more common to refer to the link loosely as "RS-232-C" or "RS-232-compatible." Please bear this in mind as you configure your interface.

The most commonly used RS-232-C lines are given in the following chart. We define the name of each line in the RS-232-C standard and the commonly used mnemonics.

The Most Common RS-232-C Circuits

The following figure depicts a typical "data link" between devices using RS-232-C.



Data Transfers

Each of the terms or conductors above has a distinct use and can be described in terms of its function within the RS-232-C protocol. We abbreviate each line or circuit for ease in description.

DESCRIPTION	ABBREVIATION (CIRCUIT)
Signal Ground	CF
Transmitted Data	XMT BB
Received Data	RCV CA
Request to Send	RTS CB
Clear to Send	CTS CC
Data Set Ready	DSR AB
Data Terminal Ready	DTR
Signal Detect	CRX CD

Transmit and Receive, for any given device, are RELATIVE TO THAT DEVICE. That is, data goes out of a device on XMT and comes in on RCV. To connect two RS-232-C devices when given the common names of the signals, you should connect XMT to RCV (in one direction) and RCV to XMT (in the other direction). If one of the devices is wired as a data-set (receive) and the other as a data-terminal (transmit), then you should connect the DTR to DTR, DSR to DSR, RTS toRTS, and so on.

The data-set/data-terminal distinction should be kept in mind because the RS-232-C interface is DIRECTIONAL. That is, each line in an RS-232-C interface has a direction--one device drives the line (sends information) and the other receives the information. Each line in an RS-232-C interface is defined as being driven by either the data-set end or the data-terminal end.

The signal ground connection (CF) is always made. RS-232-C requires that the ground potential of the two devices are connected. Devices for which this requirement cannot be met cannot and should not be connected with an RS-232-C interface.

Data Terminal Ready (DTR) is used by RS-232-C to allow the terminal to signal its readiness to send or receive data. This is a signal to modems that they have permission to answer the ringing of the telephone line.

Data Set Ready (DSR) is used by the data set to signal its readiness to send or receive data. This indicates that communications are established.

Request To Send (RTS) is used by the data terminal to tell the data set it wishes to send data. Some modems (Bell 102, for example) require this line to switch directions.

Clear to Send (CTS) allows the data set to signal its readiness to pass data from the data terminal.

The Carrier Detect (CD) line allows the data set to tell the data terminal that the communication link is established.

In normal operation, DTR, DSR, and CRX are all ON. For full duplex operations RTS and CTS are also both ON.

With full duplex operation, two devices can "handshake" with data in various ways. In full duplex mode the computer at the other end "echoes" (sends back) each character to be displayed as it is typed. This allows one to see exactly what the computer at the other end receives. It also allows the computer at the other end to decide NOT to let you see what you have typed, as in "suppressing" the echo of a password or secret code.

In half duplex operation data may pass only one direction at a time. Not all parts of the communications path need be half duplex, but if any part is, then the whole system sends data only one way at a time. In half duplex mode there is no echo back.

RS-232-C compatibility has come to cover many devices which are not "data sets" or "data terminals". This usually means that the device conforms to the ELECTRICAL RS-232-C specification. Sometimes such devices also have lines which are called DSR, DTR, RTS, and so on, but their use is often not covered by the RS-232-C standard. Usually the use is specific to the device.

RS-232-C Electrical Specifications--Voltage Levels

TYPE OF SIGNAL	FIRST STATE -24 volts to -3 volts	SECOND STATE +3 volts to +24 volts
Binary Signal	1	0
Signal Condition	MARK	SPACE
Control Function	OFF	ON

A common connector for RS-232-C is called DB-25, a "D" type. It is common practice when using this 25-pin "D" connector to connect XMT to pin 2, RCV to 3, RTS to 4, CTS to 5, DSR to 6, ground (CF) to 7, CRX to 8, and DTR to 20. However, these conventions are not always followed. You may run into cases where the other pins in the connector have either entirely unrelated functions (such as other types of communication standards on the same connector) or possibly related functions (such as setting the Baud rate by connecting two pins). READ THE INSTRUCTIONS OF ANY DEVICE YOU INTEND TO CONNECT TO THE RS-232-C INTERFACE CAREFULLY! You may have to make your own cable. Do not assume anything!

Unbelievably, the RS-232-C standard does not specify how data should be exchanged on RCV and XMT. In fact, the standard for RS-232-C avoids this issue. Fortunately, common convention and other standards have settled on a serial data transmission convention. When data is not being sent, the data line starts in an idle or the MARK state. A data character (sometimes called a transmission WORD) is signalled by asserting one START BIT, represented by the SPACE state. It is followed in time by the DATA BITS (most commonly, 8 of them), each bit being represented by SPACE for 0 and MARK for 1, at the BAUD rate. The word is terminated by 1 or sometimes 2 STOP BITS, represented by

the MARK state. The next word can immediately follow with its START BIT, and the sequence repeats. Otherwise, the line stays idle in the MARK state.

The DATA BITS are ordered least significant bit first; that is, the bit numbered 0 is sent first, 1 next, and so on. A receiver does not know when a character will be coming, so it has to constantly monitor the stopped MARK state looking for the transition to START bit. The receiver can then receive the rest of the bits in the word as it knows when each will arrive--each bit has the same duration as established by the BAUD RATE (bits-per-second-rate) of the communication. Both transmitter and receiver must use the same Baud Rate.

CONTROL AND ESC SEQUENCE CODES

Introduction

It is sometimes necessary to know control and escape sequence codes when you are installing software. The following are MBC 4000 Series codes in hexadecimal.

	CODE	HEX	FUNCTION
CTRL G	BEL	07	Ring Bell
CTRL H	BS	08	Back Space
CTRL I	HT	09	H. Tabulation
CTRL J	LF	0A	Line Feed
CTRL K	VT	0B	Cursor Up
CTRL L	FF	0C	Cursor Down
CTRL M	CR	0D	Carriage Return
CTRL X	CAN	18	Abandon ESC Sequence
CTRL Z	SUB	1A	Clear Screen
CTRL [	ESC	1B	ESC Sequence
CTRL ^	RS	1E	Cursor Home
ESC =		1B.3D	Add Cursor
ESC A		1B.41	40 Line Mode
ESC B		1B.42	Erase Line
ESC C		1B.43	Screen Copy
ESC D		1B.44	Cursor Display
ESC E		1B.45	Cursor Inhibit
ESC F		1B.46	Font Assign
ESC E		1B.47	Hard Copy
ESC H		1B.48	Cursor Home
ESC I		1B.49	Assign F Key
ESC K		1B.4B	20 Line Mode

ESC M	1B.4D	33 Line Mode
ESC N	1B.4E	Reverse Display
ESC P	1B.50	Draw Line
ESC Q	1B.51	Cursor Position
ESC R	1B.52	Console In
ESC S	1B.53	Dot Set
ESC T	1B.54	Field Attribute
ESC a	1B.61	Cursor Sense

Escape Sequence (Cursor Addressing)

ESC , " = " , ROW, COLUMN

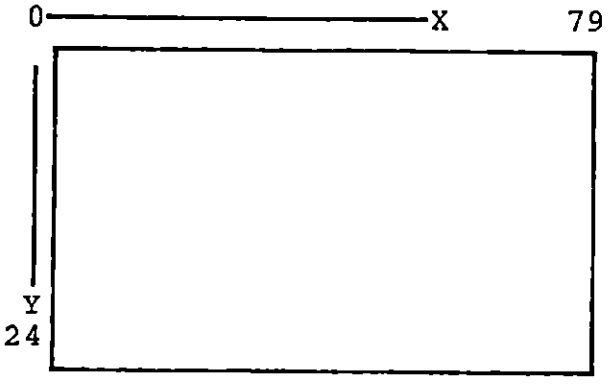
1BH 3DH 20H+Y 20H+X where Y is the column and X is the row of the screen.

Example:

X=6, Y=8

1BH, 3DH, 28H, 26H

(ESC) (=) (20H+8) (20H+6)



INDEX

A

Abbreviations in BASIC, 9-1
ABS, 10-2
AC Cable, 1-7
Arithmetic Expressions, 9-13
Arithmetic Operaotors, 9-12
Arrays, 7-6
ASC, 10-2
ASM Command, 2-8
ASM86 Command, 6-13
Assignments, 6-26
Asterisk in Commands, 6-15
Asterisk in File Names, 6-16
Asterisks in File Names, 12-3
ATN, 10-2
ATTR$, 10-2
AUTO, 9-20
AUTO Command, 11-3

B

BASIC, 9-1
    Characters and Symbols, 9-5
    Clearing a Program, 7-4
    Constants, 9-9
    Data, 9-7
    Direct Mode, 9-3
    File Names, 9-2
    Keywords and Spaces, 9-7
    Line and Statement Use, 9-5
    Line Editing, 9-4
    Lines and Line Numbers, 9-6
    Operational Modes, 9-2
    Program Edit Mode, 9-4
    Prompt and Cursor, 9-3
    Return to Operating Mode, 9-4
    Run Mode, 9-4
    Running a Program, 7-4
    Variables, 9-10
BASIC Command, 2-9
Basic Function, 10-1
BASIC History, 7-1
BASIC Programming
    Preface, 7-1
Baud Determinants
    B1 and B2, R-12
Baud Rate
    Changing, 17-10
Baud Setting

through INIT%, 9-58
BDOS Error, 6-11
BEEP, 9-21
BIOS Interface, 17-7
Blank Disks
Formatting, 2-5
Boot, 7-1
Brightness Control, 1-8
Bringing Up CP/M-86, 6-1
BS Key
Backspace Key, 2-4
Buzzer, 9-21

C

CALL, 9-22
CDBL, 10-2
CHAIN/COMMON, 9-25
Change Program Function Keys, 9-65
Changing Baud Rate, 17-10
Character Codes, R-3
CHR$, 10-2
CINT, 10-2
CLEAR, 9-28
CLOSE, 9-28
CLS, 9-31
Command and Status Formats, 17-9
Commands, 6-19
Comment Instructions, 5-1
Computer Vocabulary, 4-1
CON, 6-7
Concatenation, 8-6
Connector Pins, 17-4
Constants, 9-9
CONT, 9-32
with BREAK Key, 9-32
Control and ESC Sequence Codes, R-19
Control C, 6-30
Control S, 6-2
Control U, 6-2
Control Z, 6-34
Copy the Operating System, 13-7
Copy to Other Devices
PIP Command, 6-7
COPYDISK Command, 2-9
Copying All Files, 13-8
PIP Command, 6-6
Copying Disks, 2-5
Copying Files, 6-5
Correction of Entry, 6-2
COS, 10-2
CP/M-86
Introduction, 6-1
CP/M-86 1.1, 6-3
CPU Specifications, 17-1

CREATE, 9-33
CRT, 6-7
CSNG, 10-2
CSRLIN, 10-3
CTRL Key
   Control Key, 2-4
Cursor, 7-2
Cursor Addressing, R-20
Cursor in BASIC, 9-3
CVD, 10-3
CVI, 10-3
CVS, 10-3

C

Data
   BASIC, 9-7
Data Files, 12-4
DDT86 Command, 2-8, 6-13
DDUMP Command, 2-9, 6-13
DDUMP Utility, 13-25
Debugging, 6-14
Declarative Instructions, 5-1
DEF FN, 9-35
DEF SEG, 9-38
DEF USR, 9-39
Definitive Instructions, 5-1
DEFINT/DEFSNG/DEFDBL/DEFSTR, 9-37
DEL Key
   Delete Key, 2-4
DELETE, 9-43
Device Assignments, 6-10
Diagnostics, 14-1
DIM, 9-44
DIR Command, 2-7, 6-19
Directive Instructions, 5-1
Directory, 6-2
Disk Drive, 1-5, 2-1
Disk Files, 12-1
Disk Format, 13-5
Disk Formatting, 2-5
Disk Function, 10-1
Disk Ports, 16-1
Disk Utility Programs, 13-1
Diskette Drive Lamp, 1-5
Diskette Eject Lever, 1-5
Diskette Handling, 1-1
Diskette Interface, 15-2
Diskette Set Lever, 1-5
Disks,Copying, 2-5, 13-1, 13-24
Display, 17-1
   CRT, 1-5
Double-Precision Real Number, 9-41
Drive Change, 6-17
Drive Specification, 6-17

DSKF, 10-3

E

ED Command, 2-7, 6-12
EDIT, 9-46
Edit Keys, 2-4
Editing Commands, 6-12
END, 9-46
Ending a Program, 7-4
ENTER Key
   RETURN Key, 2-4
Entry Correction, 6-2
Environment
   Installation, 1-2
EOF, 10-3
ERA Command, 2-7, 6-20
   Erase, 6-4
ERASE, 9-47
   DIM Statement, 9-44
ERL, 10-3
ERR, 10-3
ERROR, 9-48
Error Messages, R-4
   BASIC List, R-4
   SORT Utility, 13-17
ESC Key
   Escape Key, 2-4
Escape Sequence, R-20
Evaluation of Combined Expressions, 9-18
EXP, 10-3
Exponent Sign Bit, 9-40
Expressions
   Accuracy In, 8-6
Expressions and Operators, 9-11
Extensions, 6-4
External Storage, 17-1

F

FDUMP Command, 2-9
FDUMP Utility, 13-21
FG Terminal, 1-7
FIELD, 9-49
File Copying, 2-7
File Name or Command
   Typing In, 6-2
File Names, 6-3, 12-1
   Extention, 12-1
   Qualifiers, 12-1
   Sample, 6-15
File Names in BASIC, 9-2
File Number
   Allowable Per Diskette, 12-10
File Referencing, 6-16

File Transfer Program, 13-3
File Types
   BASIC, 12-1
Files
   Number of, 12-3
Files with Commands, 6-15
FILES/LFILES, 9-50
FIX(X), 10-4
Floppy Disk Interface, 15-2
Flow, 5-2
FN, 10-8
FOR/NEXT, 9-51
FORMAT Command, 2-9, 6-21
Formats
   Command and Status, 17-9
Formatting a Diskette, 13-5
FRE, 10-4
Functions, 10-1
Fuses, 1-7

G

GENCMD Command, 2-9
GENDEF Command, 2-9
GET, 9-53
   with PUT Statement, 9-53
Glossary, 4-1
GOSUB Command, 7-7
GOSUB/RETURN, 9-54
GOTO, 9-56
Graph Key, 2-3

H

Hardware Configuration, 17-3
Hardware Specifications, 17-1
HCOPY, 9-56
HELP Command, 2-9
HEX, 10-4

I

I/O Map, 17-6
I/O Structures, R-9
IF/THEN/ELSE, 9-57
Imtermediate Files, 6-3
Index Sequential File
   PUT Statement, 9-98
   record pointer
      SETKEY, 9-110
Index Sequential File Allocation, 9-33
INIT%, 9-58
Initialization
   Sanyo BASIC, 9-2
INKEY$, 10-4

INP, 10-4
INPUT, 9-60
Input and Output of Files and Disk Images, 12-9
INPUT#, 9-62
INPUT$, 10-4
INPUT%, 9-63
Input/Output Firmwares, R-9
Input/Output Software Awareness, 16-1
Input/Output Systems and Connectors, 15-1
Input/Process/Store/Output, 3-1
INSTR, 10-4
Instruction Words, 9-19
Instructions for Files, 12-3
INT, 10-4
Interface, 17-2

J

Jumpers, R-11

E

KEY, 9-64
KEY LIST
   Program Function Keys
      Changing, 9-65
KEY LIST/KEY LLIST, 9-65
KEY LOAD/KEY SAVE, 9-65
Keyboard, 1-5, 2-2, 15-2
Keyboard Connector, 1-7
Keyboard Controls, 16-1
Keyboard Specifications, 17-1
KILL, 9-66
Kill Command, 6-18

L

LDCOPY Command, 2-7, 6-9
LEFT$, 10-5
LEN, 10-5
Length Bits, R-12
LET, 9-67
LF Key
   Line Feed Key, 2-4
Line Editing, 11-1
LINE INPUT, 9-68
LINE INPUT#, 9-70
LINE INPUT%, 9-71
Line Numbers, 9-20
   BASIC, 9-6
LIST Command
   BASIC, 7-3
LIST/LLIST, 9-72
LMCMD Command, 2-9
LOAD, 9-74

LOADER Command, 2-9
LOC, 10-5
LOCATE, 9-75
LOF, 10-5
LOG, 10-5
Logical Device Names, 6-25
Logical Expression, 9-15
Logical Operators, 8-4, 9-15
LPOS, 10-5
LPRINT/LPRINT USING, 9-76
LPT, 6-7
LSET/RSET, 9-76
LST, 6-7

M

Mantissa Sign Bit, 9-40
Math and Logical Relationships, 8-1
Mathematical Operators, 8-1
Memory, 5-2
Memory Map for CPU, 17-5
Memory Maps, R-8
Memory Specifications, 17-1
MERGE, 9-78
MERGE Statement, 11-2
Merge Utility, 13-18
Merging Files Together
   PIP Command, 6-6
MID, 9-79
MID$, 10-5
MKD$, 10-5
MKI$, 10-6
MKS$, 10-6
MOVCPM Command, 2-9, 6-15
MRCV Command, 2-9
MTRN and MRCV Programs, 13-4
MTRN Command, 2-9

N

NAME, 9-81
Names of Parts
   Functions of Parts, 1-4
NEW, 9-81
Number of Files on Disk, 6-18
Numeric Function, 10-1
Numeric Keys, 2-4

O

OCT$, 10-6
ON ERROR GOTO/RESUME, 9-82
ON GOSUB/ON GOTO, 9-84
OPEN, 9-86
Operating System, 5-2

Running, 2-7
Operating System Transfer, 13-7
Operation, 2-1
Operation Modes of SANYO BASIC, 9-2
Operation Sequence, 13-10
Operators Onto Strings, 8-6
OPTION BASE, 9-88
OUT, 9-89

P

PACK$, 10-6
Parentheses in Expressions, 8-2
PEEK, 10-6
PF Keys
Change, 9-65
Physical Device Names, 6-25
PIP Command, 2-8, 6-5, 6-31
POKE, 9-90
POS, 10-6
Power, 1-5
PRINT, 9-91
Print Function, 10-1
PRINT USING, 9-92
PRINT#/PRINT# USING, 9-96
PRINT%/PRINT% USING, 9-97
Printer
Software Control, 16-1
Printer Connection, 1-3
Printer Connector, 1-7, 15-1
Printer Interaction, 6-5
Printer Pins, 15-1
Program Deletion
SANYO BASIC, 11-2
Program Editing, 11-1
Program Editor
SANYO BASIC, 11-1
Program Execution Transfer, 7-7
Program Files, 12-4
Program Function Keys
Change, 9-65
Program Listing
SANYO BASIC, 11-2
Program Merging
SANYO BASIC, 11-2
Program Saving
SANYO BASIC, 11-2
Programmable Keys, 2-3, 15-2
Changing, 9-65
Programming and Flow Relationships
Operating System, 5-1
Prompt in BASIC, 9-3
PTP, 6-7
PTR, 6-7
PUN, 6-7

PUT, 9-98
PUT Command, 2-9

Q

Qualifier in File Names, 12-1
Question Mark in File Name, 6-4
Question Marks in File Names, 6-16

R

RAM, 17-1
Random Files, 12-7, 12-10
RANDOMIZE, 9-100
RDR, 6-7
Read Bit Modes, R-12
READ/DATA/RESTORE, 9-101
Recommended Diskettes
   Diskettes
      Recommended, 1-1
Relational Expression, 9-14
Relational Operators, 9-13
REM, 9-103
REN Command, 2-7, 6-22, 11-3
   Rename, 6-4
RENUM, 9-104
Repeat, 2-3
Reserved Words, R-2
RESET, 9-105
Reset Key, 2-4
Resident Commands, 6-19
Resident CP/M Commands, 2-7
RESUME, 9-82, 9-106
RETURN Key
   CR Key, 2-4
RIGHT$, 10-6
RND, 10-7
ROM, 17-1
RS-232 Electrical Specifications, R-17
RS-232 Ports
   Use, R-11
RS-232C Port Configuration Control, 15-3
RS232 Input/Output Serial Port, R-14
RS232 Program, 17-11
RS232 Serial Port
   Use of, 17-8
RUN, 9-107
Run Command, 7-4

S

SANYO BASIC, 9-1
SANYO BASIC Instructions and Forms, 5-1
SANYO BASIC Line Length, 11-1

SAVE, 9-108
SAVE Command, 6-5, 6-29
Segment Base Address, 9-38
Sequence of Operations, 13-10
Sequential Files, 12-4, 12-6, 12-9
Serial Interface
   RS-232C, 1-7
Serial Port, 15-1
Serial Ports
   Software Control, 16-1
SET, 9-109
SETKEY, 9-110
SGN, 10-7
Shift Key, 2-3
SIN, 10-7
Small Key, 2-3
Software Specifications, 17-2
SORT Command, 2-9
Sort Utility, 13-9
   Limitations, 13-10
SPC, 10-7
Special Parameters During Copy and Transfer, 6-8
Specifications, 17-1
SQR, 10-7
Stack Status, 9-23
STAT Command, 2-9, 6-10, 6-23
   Contents of an Extention, 6-18
Status and Command Formats, 17-9
STOP, 9-110
Stop Bits, R-13
Store Function, 3-2
STR, 10-7
String Expression, 9-19
String Function, 10-1
STRING$, 10-7
String-numeric Function, 10-1
SUBMIT Command, 2-8, 6-15, 6-27
SWAP, 9-110
Syntax, 5-1, 7-3, 9-1
SYSGEN, 13-7
SYSTEM, 9-111
System Boot, 5-2
System Connection, 1-2

T

TAB, 10-8
Tab Key, 2-4
TAN, 10-8
TINPUT, 9-111
TOD Command, 2-9
Transfer Instructions, 5-1
Transfer Parameters, 13-3

Transient CP/M Commands, 2-7
TRON/TROFF, 9-114
Trouble-shooting, 17-12
Trouble-shooting List and Remedies, 14-1
TTY, 6-7
TYPE Command, 2-7, 6-4, 6-30
Typing in a File Name or Command, 6-2

U

UART, R-11
UC1, 6-7
UL1, 6-7
UNPACK$, 10-8
UP1, 6-7
UR1, 6-7
USR, 10-9

V

V-HOLD Control, 1-8
VAL, 10-8
variable
   definition statement, 9-37
Variables, 7-5, 9-10
   Numeric, 7-5
VARPTR, 10-8
Ventilation Slots, 1-8

W

WAIT, 9-115
WHILE/WEND, 9-116
WIDTH, 9-118
Withdrawing the Diskette, 2-1
Working With Drive B, 6-1
WRITE, 9-120
Write Bit Modes, R-12

X

XCHG Utility, 13-23
XREF Utility, 13-22

Printed in Japan Apr./83 KW
9376411909501

NOTICE

When powered on, some meaningless characters may appear on the screen. Soon the screen returns to normal since the boot program on the disk is loaded on the moemory(cold start). This is the normal phenomenon of the CRT controller used in the machine and nothing wrong with the machine.